VoIP FeTAp – Internet-Telefonie mit einem alten Wählscheibenapparat

Diplomarbeit im Studiengang Multimedia
Fachbereich Informatik
Fachhochschule Augsburg

Doris Schöllhorn
Matrikelnummer: 902812
Sommersemester 2006

Erstprüfer: Prof. Dr. Wolfgang Kowarschick
Zweitprüfer: Prof. Dr. Hubert Högl

Erstellungserklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und anderweitig noch nicht für Prüfungszwecke vorgelegt habe. Es wurden ausschließlich die angegebenen Quellen oder Hilfsmittel benutzt. Wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Augsburg, 19. April 2006

_____

Doris Schöllhorn

Inhaltsverzeichnis

1. Einleitung 6
1.1. Motivation 6
1.2. Aufgabenstellung 7
1.3. Gliederung 7

2. Wichtige Komponenten 9
2.1. Wählscheibentelefon 9
2.1.1. Telefonsignaländerung beim Anrufen 10
2.1.2. Telefonsignaländerung beim Angerufenwerden 11
2.2. Mikrocontroller 11
2.2.1. Auswahl Mikrocontroller-Modell 12
2.2.2. ATmega8 in Kürze 12
2.2.2. Bauform 13
2.2.3. Digitale bidirektionale I/O-Ports 14
2.2.4. Spannung 14
2.2.5. Stromverbrauch 14
2.2.6. Takt/Geschwindigkeit 14
2.2.7. Speicher 15
2.2.8. Peripheriefunktionen 15
2.2.9. Programmiersprache 15
2.2.10. Programmübertragung 15

3. Elektronische Schaltung 16
3.1. Lochrasterplatine 1 16
3.1.1. ATmega8 anschließen 20
3.1.2. Spannungsversorgung 20
3.1.3. Reset-Schalter 20
3.1.4. Externer Taktgeber 21
3.1.5. Betriebsspannung für Analog-Digital-Wandler 21
3.1.6. Mikrofon-Schaltung 21
3.1.7. Lautsprecher-Schaltung 23
3.1.8. SPI-Schnittstelle 24
3.1.9. RS232-Schnittstelle 25
3.1.10. USB-Schnittstelle 26
3.1.11. Verbindung mit Platine 2 29
3.2. Lochrasterplatine 2 29
3.2.1. 9V Gleichspannung 32
3.2.2. Spannungsregler 33
3.2.3. Schutzwiderstand 33
3.2.4. 5V-Relais 33

3.2.5. 40V Wechselspannung . . . 34
3.2.6. Telefonanschluss . . . 34
3.2.7. Telefonsignalverarbeitung . . . 34
3.2.8. Operationsverstärker . . . 34
3.2.9. Wählimpuls-Schaltung . . . 34
3.2.10. Telefonhörerstatus-Schaltung . . . 36
3.2.11. Sprachsignal-Schaltung . . . 39
3.3. Verbindung der zwei Lochrasterplatinen . . . 40

4. Verwendete Programme . . . 41
4.1. WinAVR . . . 41
4.1.1. AVRDude 4.4.0 . . . 41
4.1.2. Die Datei install_giveio.bat . . . 42
4.3. Cygwin . . . 42
4.4. HyperTerminal . . . 43
4.5. MProg . . . 43

5. Software . . . 44
5.1. Programmiersprache . . . 44
5.2. Compiler . . . 44
5.3. Entwicklungsumgebung . . . 45
5.4. Bibliotheken . . . 45
5.4.1. AVR-Libc . . . 45
5.4.2. Procyon AVRlib . . . 46
5.4.3. Verwendete Dateien der Procyon AVRlib . . . 46
5.5. Header-Datei global.h . . . 47
5.6. Erzeugen von Maschinencode . . . 47
5.7. Makefile . . . 48
5.8. Fuse-Bits programmieren . . . 48
5.8.1. Externen Quarz verwenden . . . 48
5.8.2. Fuse-Bits abfragen und programmieren . . . 49
5.9. Timer . . . 51
5.10. Vorteiler . . . 51
5.11. Interruptgesteuerter Programmablauf . . . 51
5.11.1. In der Software verwendete Interrupts . . . 52
5.11.1.1. Überlauf Timer 0 . . . 52
5.11.1.2. Externe Interrupts . . . 52
5.12. Pollingverfahren . . . 52
5.13. Watchdog-Timer . . . 53
5.14. C-Dateien des Projekts . . . 53
5.14.1. Datei cmdlinetest.c . . . 53
5.14.1.1. Bibliotheken einbinden . . . 53
5.14.1.2. Interrupts deaktivieren . . . 54

5.14.1.3. Initialisierung 54
5.14.1.4. Datenrichtung bestimmen 55
5.14.1.5. Funktion goCmdline() 57
5.14.2. Datei timer.c 59
5.14.3. Analog-Digital-Wandler 63
5.14.4. Datei a2d.c 63
5.14.5. Pulsweitenmodulation 65
5.14.5.1.Genaue Erklärung der Hardware-Pulsweitenmodulation 66
5.14.6. Datei pwmsw.c 69
5.14.7. Datei teltools.c 70
5.15. Batch-Datei 78

6. VoIP FeTAp in der Praxis 80
6.1. Ablaufprotokoll beim Anrufen 80
6.2. Ablaufprotokoll beim Angerufenwerden 81

7. Überlegungen zur Weiterentwicklung 83
7.1. Treiber für PC 83
7.2. Soundbibliothek 83
7.3. VoIP 83
7.3.1. Protokoll H.323 84
7.3.2. Protokoll SIP 84
7.3.3. VoIP-Software 84
7.3.4. Datentransport 85
7.3.5. Nach dem Datentransport 85

8. Fazit 86

A Beigefügte CD-ROM 87
A.1. Inhalt der beigefügten CD-ROM 87
A.2. CD-ROM 88

B Lizenz 89

C Abbildungsverzeichnis 91

D Glossar 92

E Quellenverzeichnis / Literaturverzeichnis 96

1. Einleitung

1.1. Motivation

Die Motivation zu dieser Arbeit war, alte und neue Technik miteinander zu vereinen. Ein altes Wählscheibentelefon mit einem modernen Mikrocontroller so zu verbinden, dass es möglich wird die neueste Technik im Telefonie-Bereich – „Voice over IP“ (VoIP) – verwenden zu können.

Warum ist VoIP so interessant? VoIP ist stark im Kommen, nicht zuletzt deswegen, weil es bereits von vielen Firmen für den Privatanwender als kostengünstige alternative angeboten wird. 14 Prozent des deutschen Mittelstandes nutzen bereits Internet-Telefonie [VoIPinfo (10.02.06)], Tendenz steigend.
Für den Anwender ist VoIP sehr kostengünstig und ohne weiteren Aufwand mit dem vorhandenen Telefon und einem Internetanschluss praktizierbar.

Auch die deutsche Telekom hat angekündigt, das herkömmliche Festnetz bis spätestens zum Jahr 2012 vollständig durch IP-basierte Technologie abzulösen. [Telt]

Motiviert hat mich außerdem, mich mit den – im Alltag in fast jedem Gerät versteckten – Mikrocontrollern zu beschäftigen.

Der immerwährende Trend zur Nostalgie hat mich auch ermutigt, mich mit der „alten“ Technik – dem Wählscheibentelefon (früher „Fernsprechtischapparat“ genannt, kurz „FeTAp“) – auseinander zusetzen. Die Verbindung von neuer und alter Technik spiegelt sich auch im Projektnamen „VoIP FeTAp“ wieder.

1.2. Aufgabenstellung

Ziel dieser Arbeit war es zunächst ein Hardware-Interface zwischen einem Wählscheibentelefon und einem PC, der unter „Microsoft Windows XP“ läuft, zu entwickeln.

Im Hardware-Interface befindet sich ein Mikrocontroller, der hier als „Gehirn“ fungiert, denn er muss die Signale des Wählscheibentelefons verstehen und an den PC weiterleiten können. Auch Signale vom PC sollen über den Mikrocontroller Auswirkungen auf das Wählscheibentelefon haben.

Die Aufgabe war es nun, die Signale des Wählscheibentelefons so aufzubereiten, dass sie für den Mikrocontroller leicht weiterzuverarbeiten sind.

Der Mikrocontroller soll über USB an den PC gekoppelt werden.

Außerdem soll statt einer Telefonnummer die IP-Adresse des Gesprächspartners gewählt werden können.

1.3. Gliederung

Diese Arbeit beschreibt alle verwendete Hardware und Software. Auf Grund dessen ist es möglich, das Produkt dieser Arbeit selbst nachzubauen.
Vorausgesetzt werden allerdings grundlegende Unix-Kenntnisse und Kenntnisse der Programmiersprache C. Außerdem ist diese Arbeit an einem PC mit dem Betriebssystem „Microsoft Windows XP“ erstellt worden und geht bei der Erläuterung auch nicht auf andere Betriebssysteme ein.

Diese Arbeit ist in 8 Kapitel gegliedert, die sich wie folgt zusammenfassen lassen:

Nach der Einleitung werden die zwei für die Realisierung der Hardware wichtigsten Komponenten in Kapitel 2 beschrieben. Kapitel 3 beschäftigt sich dann mit den aus dieser Arbeit entstandenen Lochrasterplatinen und beschreibt deren Aufbau und Funktion im Detail. Des Weiteren werden in Kapitel 4 alle verwendeten Programme näher erklärt. Die entstandene Software wird in Kapitel 5 ausführlich dokumentiert. Kapitel 6 beschreibt die Ablaufprotokolle beim Anrufen und Angerufenwerden ein abschließendes Mal, um die genaue Funktionsweise des Hardware-Interfaces verständlich zu machen. In Kapitel 7 werden Überlegungen angestellt, wie diese Arbeit weiterentwickelt werden könnte. Kapitel 8 stellt das Fazit dar.

Folgende Schriftkonventionen kamen in dieser Arbeit zum Einsatz:

`feste Breite`	wird verwendet für Quelltexte
kursiv	wird verwendet für Begriffe, die im Glossar erklärt werden
fett	wird verwendet für Hervorhebungen

Sinngemäße Zitate und Quellen sind mit „[AUTOR oder QUELLE]", direkt dem Satz oder Absatz folgend, gekennzeichnet. In den eckigen Klammern steht die Kurzbezeichnung des Autors oder der Quelle und ist damit im Quellenverzeichnis zu finden.

Bei Abgabe dieser Arbeit sind alle Internetverweise öffentlich zugänglich.

2. Wichtige Komponenten

2.1. Wählscheibentelefon

Verwendet wurde ein altes Wählscheibentelefon aus dem Jahr 1986. Genaue Typbezeichnung dieses Wählscheibentelefons: „FeTAp 791-1“.

Abbildung 2.1.: Das Wählscheibentelefon

Grundsätzlich benötigt ein Telefon eine Versorgungsspannung von ca. 10 Volt Gleichspannung, die es für gewöhnlich vom Amt bekommt. In meiner Arbeit habe ich hierfür eine externe Spannungsquelle angeschlossen, die konstant 9 Volt Gleichspannung liefert.

Das Telefon hat vier ausgehende Leitungsadern, von denen aber zur normalen Telekommunikation nur zwei verwendet werden. Über diese zwei Leitungen werden die Sprache, die Wählsignale und das Signal zum Klingeln des Telefons übertragen.

2.1.1. Telefonsignaländerung beim Anrufen

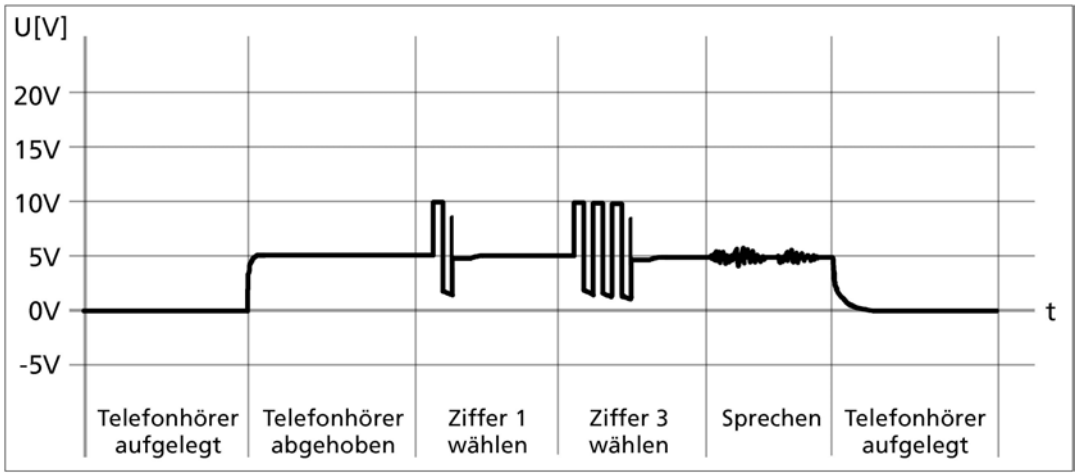


Abbildung 2.2.: Telefonsignaländerung beim Anrufen

In dieser Abbildung ist gut erkennbar, wie sich das Signal in der Leitung – beim Wunsch einen Gesprächspartner anzurufen – verändert. Befindet sich der Telefonhörer noch auf der Telefongabel, dann liegt das Telefonsignal bei 0V.

Beim Abheben des Telefonhörers wird im Telefon eine Stromschleife geschlossen, wodurch das Signal mit etwas Verzögerung auf 5V steigt. Somit kann das Abheben durch die Signaländerung leicht erkannt werden.

Beim Wählen einer Ziffer tritt das Impulswahlverfahren in Kraft. Dieses Verfahren benutzen auch heute noch viele ältere Telefongeräte.
Dreht man nun die Wählscheibe, um eine Ziffer zu wählen, steigt das Signal von 5V auf 10V an. Beim Wählen der Ziffer 3 sinkt das Signal drei mal für eine bestimmte Zeitdauer auf 0V, steigt jeweils danach drei mal kurz auf 10V an und bleibt dann, wenn die Wählscheibe wieder in Ruheposition ist, bei 5V. Diese eindeutigen Signaländerungen gewährleisten ein einfaches Zählen der Wählimpulse.

Das analoge Sprachsignal bewegt sich in einem Bereich zwischen 500mV und 1V und liegt im Bereich um die 5V.

Beim Ablegen des Telefonhörers nach dem Telefonat sinkt das Signal mit etwas Verzögerung wieder auf 0V ab.

2.1.2. Telefonsignaländerung beim Angerufenwerden

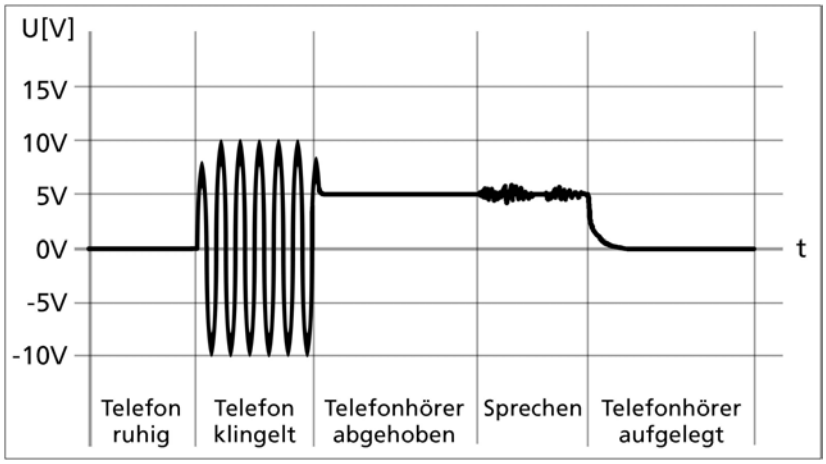


Abbildung 2.3.: Telefonsignaländerung beim Angerufenwerden

Wird man angerufen, so wird die Telefonklingel normalerweise vom Amt mit 60V Wechselspannung versorgt. Da auch 40V ausreichen, wird die Telefonklingel bei dieser Arbeit mit einem 40V Transformator betrieben. Diese 40V Wechselspannung legt sich auf die 9V Versorgungsspannung und ergibt das Signal, wie es in Abbildung 2.3. ersichtlich ist.

2.2. Mikrocontroller

In dieser Arbeit hat der Mikrocontroller eine zentrale Rolle. Er stellt sozusagen das „Gehirn“ dar. Er steht zwischen Telefon und PC und ermöglicht die Kommunikation zwischen beiden.

Ein Mikrocontroller ist im Grunde ein (fast) kompletter Computer auf einem einzigen Chip. Die CPU steckt – ganz anders als bei einem gewöhnlichen PC – zusammen mit der gesamten I/O-Elektronik und dem Speicher auf einem Halbleiter. In punkto Geschwindigkeit kommen Mikrocontroller mit modernen CPU’s für PC’s bei weitem nicht mit. Dafür sind sie bezüglich der Leistung auch sehr viel genügsamer und kommen ohne spezielle Lüfter und Kühlungsmaßnahmen aus.

Die einzelnen Mikrocontroller werden an der Bit-Zahl des internen Datenbusses unterschieden. Es gibt 4-bit, 8-Bit, 16-bit und 32-bit. Diese Bit-Zahl kann als die Länge der Daten interpretiert werden, die der Mikrocontroller in einem Befehl verarbeiten kann.

2.2.1. Auswahl Mikrocontroller-Modell

Der Mikrocontroller sollte für diese Arbeit idealerweise folgende Voraussetzungen erfüllen:

- Gute Beschaffbarkeit und geringer Preis
- Handliche Bauform (ein Mikrocontroller mit zu vielen Pins („Beinchen“) ist schwieriger zu handhaben)
- Flash-ROM (der Mikrocontroller sollte mindestens 1000 mal neu programmiert werden können)
- In-System-Programmierbarkeit (ISP) (man benötigt hier kein teures Programmiergerät und muss den Controller zur Programmierung nicht aus der Schaltung entfernen)
- Kostenlose Software verfügbar

Zur Zeit werden diese Anforderungen sehr gut von den 8-bit-AVR-Mikrocontrollern der amerikanischen Halbleiter-Firma „Atmel“ erfüllt.

Diese Mikrocontroller mit RISC-Architektur besitzen eine zweistufige Pipeline, die es ermöglicht, die meisten Befehle innerhalb eines einzigen Prozessortaktes auszuführen. Die Mikrocontroller von Atmel haben gegenüber einer „normalen“ CPU 32 x 8 Register anstatt 4. Diese 32 Allzweckregister sind für den allgemeinen Gebrauch. Sie sind alle direkt mit der *Arithmetischen-Logischen-Einheit (ALU)* verbunden, was erlaubt, mit einer einzigen Anweisung auf zwei unabhängige Register zuzugreifen und das in nur einem Taktzyklus. Die sich daraus ergebende Architektur ist code-effizienter indem sie einen zehn mal schnelleren Durchsatz erzielt als konventionelle CISC-Mikrocontroller.

Für diese Arbeit wurde der 8-bit-AVR-Mikrocontroller ATmega8 von Atmel verwendet, der im Folgenden Text kurz „ATmega8“ genannt wird.

2.2.2. ATmega8 in Kürze

- Genaue Bezeichnung: ATMEGA8-16PI
- Gehäuseform: PDIP28
- 8192 Byte Flash Programmspeicher
- 1024 Byte RAM
- 512 Byte EEprom
- 23 I/O’s: bidirektional

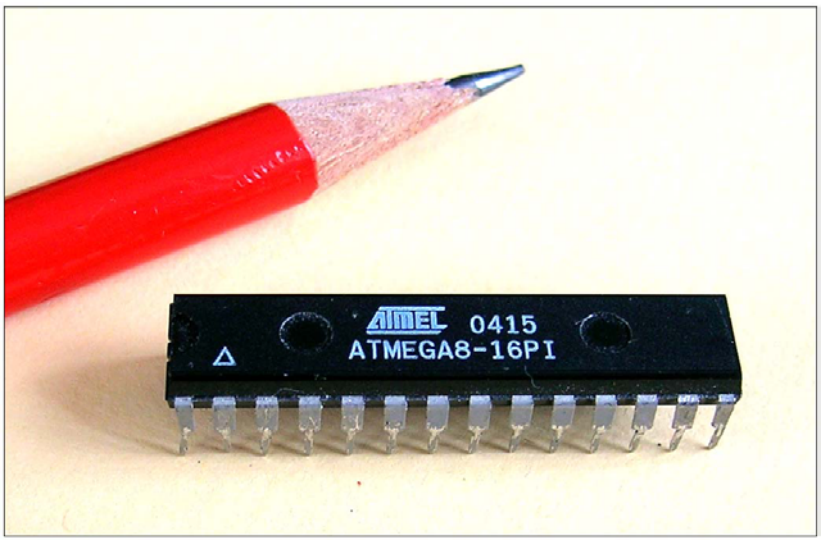


Abbildung 2.4.: Der Mikrocontroller ATmega8 von der Firma Atmel

2.2.2. Bauform

Den ATmega8 gibt es in verschiedenen Bauformen. In dieser Arbeit habe ich mich für die klassische Bauform PDIP28 entschieden. In dieser Bauform besitzt der ATmega8 28 Pins („Beinchen").

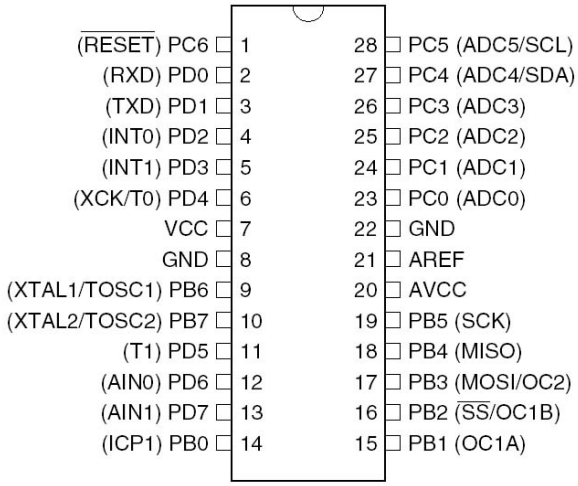


Abbildung 2.5.: Pin Konfiguration des ATmega8 [ATmega8]

2.2.3. Digitale bidirektionale I/O-Ports

Beim ATmega8 gibt es 3 Ports (B-D), denen die Pins zugehören. Jeder Pin kann individuell als Eingang oder Ausgang konfiguriert werden. Die meisten Ports sind doppelt belegt, sie besitzen neben der normalen Port-Funktion noch eine Sonderfunktion.

An jedem Pin gibt es zuschaltbare interne *Pull-up-Widerstände*, die teilweise auch bei aktivierter Sonderfunktion verfügbar sind. Möchte man, dass der ATmega8 an einem Pin die Versorgungsspannung (VCC) anlegt, so muss man lediglich den Pull-up-Widerstand einschalten.

2.2.4. Spannung

Der ATmega8 benötigt zum Betrieb eine Versorgungsspannung (VCC) von 4,5 bis 5,5 Volt. Netzteile liefern im Leerlauf manchmal deutlich mehr Spannung als angegeben, ein Spannungsregler ist deshalb Pflicht.

2.2.5. Stromverbrauch

Im Vergleich zu PC-Prozessoren (Pentium, Athlon usw.) brauchen Mikrocontroller relativ wenig Strom. Der Stromverbrauch des ATmega8 hängt stark vom Takt ab.

Takt	Stromverbrauch
32 kHz	80µA
100 kHz	0,5mA
1 MHz	2,3mA
8 MHz	11mA
16 MHz	20mA

Tabelle 2.1.: Stromverbrauch des ATmega8

2.2.6. Takt/Geschwindigkeit

Der ATmega8 wird mit einem internen Taktgeber von 1 MHz ausgeliefert. Mit einem externen Taktgeber ist es jedoch möglich einen Takt von bis zu 16 MHz zu erreichen.

2.2.7. Speicher

Das Programm, das der ATmega8 ausführt, wird im 8192 Byte großen Flash-Speicher abgelegt. Im 1024 Byte großen SRAM werden die Variablen, der Stack usw. gespeichert. Das SRAM kann beliebig oft wiederbeschrieben werden, allerdings sind die Daten nach dem Abschalten weg. Im 512 Byte großen EEprom können Werte gespeichert werden, die man dauerhaft speichern möchte, allerdings kann das EEprom nur 100 000 mal beschrieben werden.

2.2.8. Peripheriefunktionen

Der ATmega8 hat eine ganze Menge Peripheriefunktionen hardwaremäßig implementiert, z.B.: UART, I2C-Bus, SPI, PWM usw. Der Vorteil liegt darin, dass der Mikrocontroller damit mehrere Anfragen gleichzeitig ausführen kann. Dadurch steigt zum einen die Gesamtleistung des Mikrocontrollers, zum anderen sind viele Dinge zeitkritisch, und die Programmierung ist deutlich einfacher, wenn man zehn zeitkritische Anfragen gleichzeitig erledigen kann.

2.2.9. Programmiersprache

Den besten Zugriff auf die „Innereien“ des Mikrocontrollers hat man mit Assembler. Bei Zunahme des Umfangs und der Komplexität von Assembler Programmen wächst aber der Wunsch nach einer effektiveren Programmierumgebung.
Für größere Entwicklungen werden somit mittlerweile auch höhere Programmiersprachen eingesetzt. Hier ist die Programmiersprache „C“ klar dominierend. Ein Vorteil der höheren Programmiersprachen gegenüber Assembler ist z. B., dass diese Programme portabel sind, d. h. eine einmal erstellte Programmstruktur lässt sich auf andere Mikrocontroller-Typen übertragen.

2.2.10. Programmübertragung

Der Controller verfügt über eine sogenannte SPI-Schnittstelle. Das bedeutet, dass der ATmega8 über bestimmte Pins mit einer geeigneten PC-Software programmiert werden kann. Zum Anschluss an den PC benötigt man einen ISP-Programmieradapter. Hier gibt es verschiedene Lösungen, serielle und parallele Adapter, original von „Atmel“ oder kompatible Lösungen. Der in dieser Arbeit verwendete Programmieradapter wird am Druckerport betrieben.

3. Elektronische Schaltung

Bei den elektronischen Schaltungen wurden nur handelsübliche und leicht zu beschaffende Bauteile verwendet.

Für diese Arbeit wurden zwei Lochrasterplatinen eingesetzt, mit denen man schnell funktionsfähige Schaltungen aufbauen kann, die auch leicht wieder verändert werden können. Die zwei Lochrasterplatinen haben eine Abmessung von 10 x 16 cm und ein Raster von 1 mm Bohrungen mit einem Lochrasterabstand von 2,54 mm.

Die erste Lochrasterplatine wurde mit dem ATmega8 und den benötigten Teilen für die Kommunikation zwischen PC und ATmega8 bestückt. Zusätzlich befinden sich auf der ersten Lochrasterplatine noch Filter für den Audioaus und -eingang.

Auf der zweiten Lochrasterplatine befindet sich die Schaltung für die Aufbereitung der Telefonsignale für den ATmega8.

Im folgenden werden die Bestückungen der beiden Lochrasterplatinen einzeln erklärt.

3.1. Lochrasterplatine 1

Genauer befinden sich auf Platine 1 der ATmega8, ein MAX232 Pegelwandler, der die serielle Kommunikation zwischen PC und ATmega8 realisiert, ein DLP2232M FTDI-Modul, das zusätzlich die Kommunikation zwischen PC und ATMEGA8 über USB ermöglicht und eine SPI-Schnittstelle, die die In-System-Programmierung (ISP) unterstützt. Außerdem befinden sich auf dieser Platine noch einige *Operationsverstärker*, die bei der Aufbereitung vom Sprachsignal behilflich sind. Die gesamte Lochrasterplatine wird mit einer 5V Gleichspannung betrieben. Diese stabilen 5V werden auf der zweiten Lochrasterplatine durch einen Spannungsregler erzeugt. Im Folgenden wird genau erläutert, wie diese einzelnen Bauteile auf Platine 1 miteinander verbunden sind. Allen Bauteilen wurde eine eindeutige Bezeichnung gegeben, die bei der Erklärung immer, eventuell auch in Klammern, mitangegeben ist.

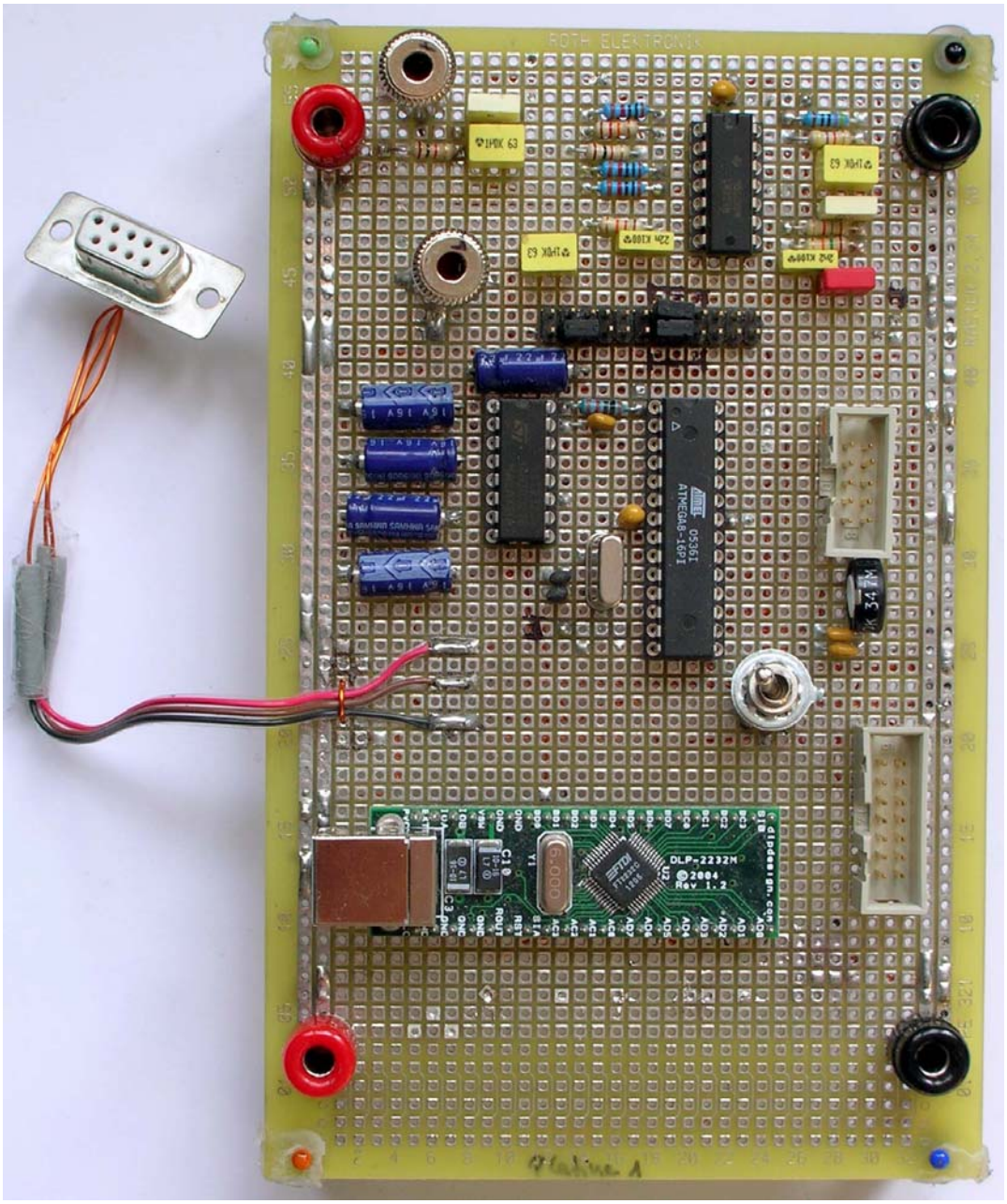


Abbildung 3.1.: Lochrasterplatine 1

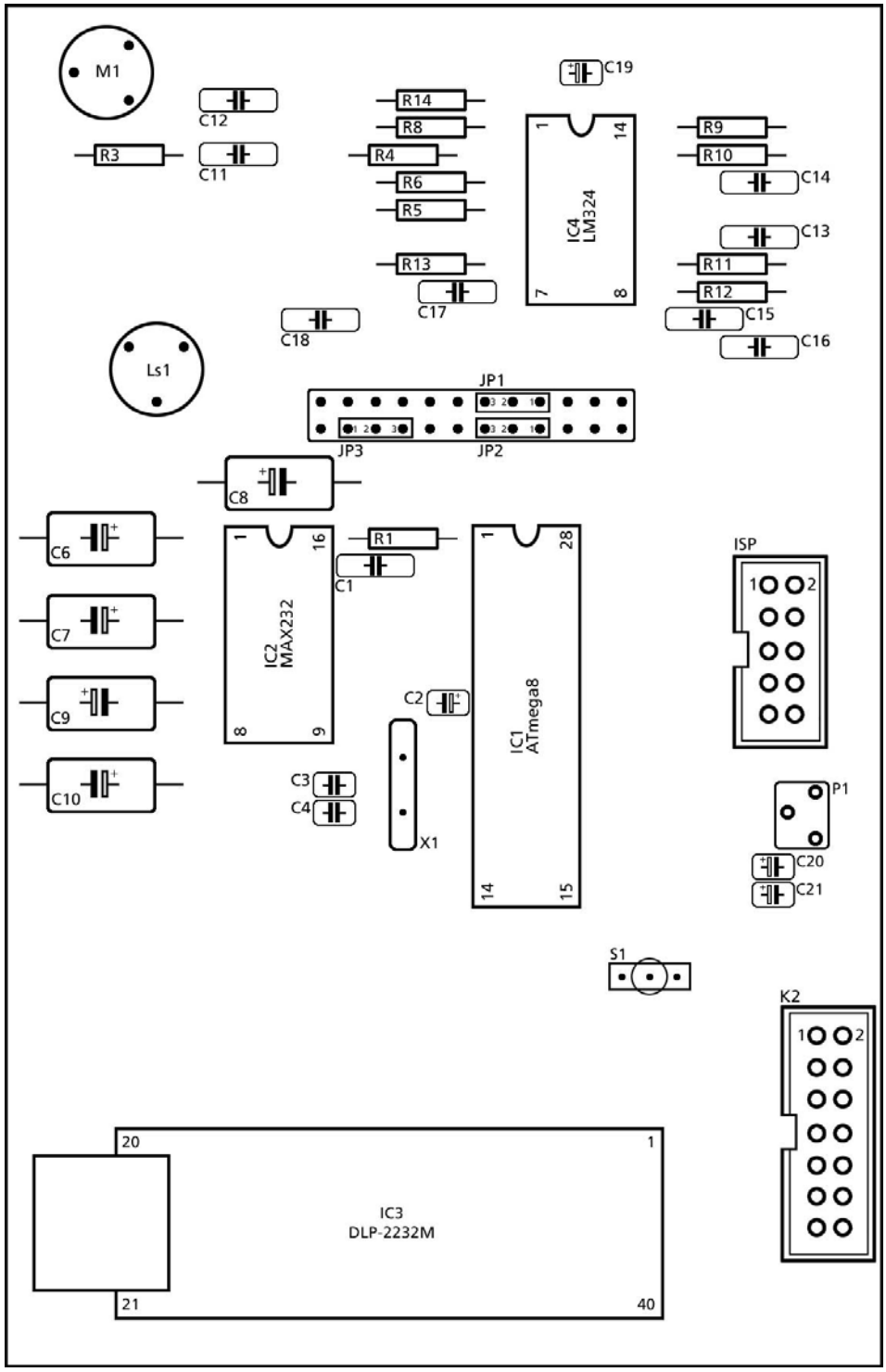
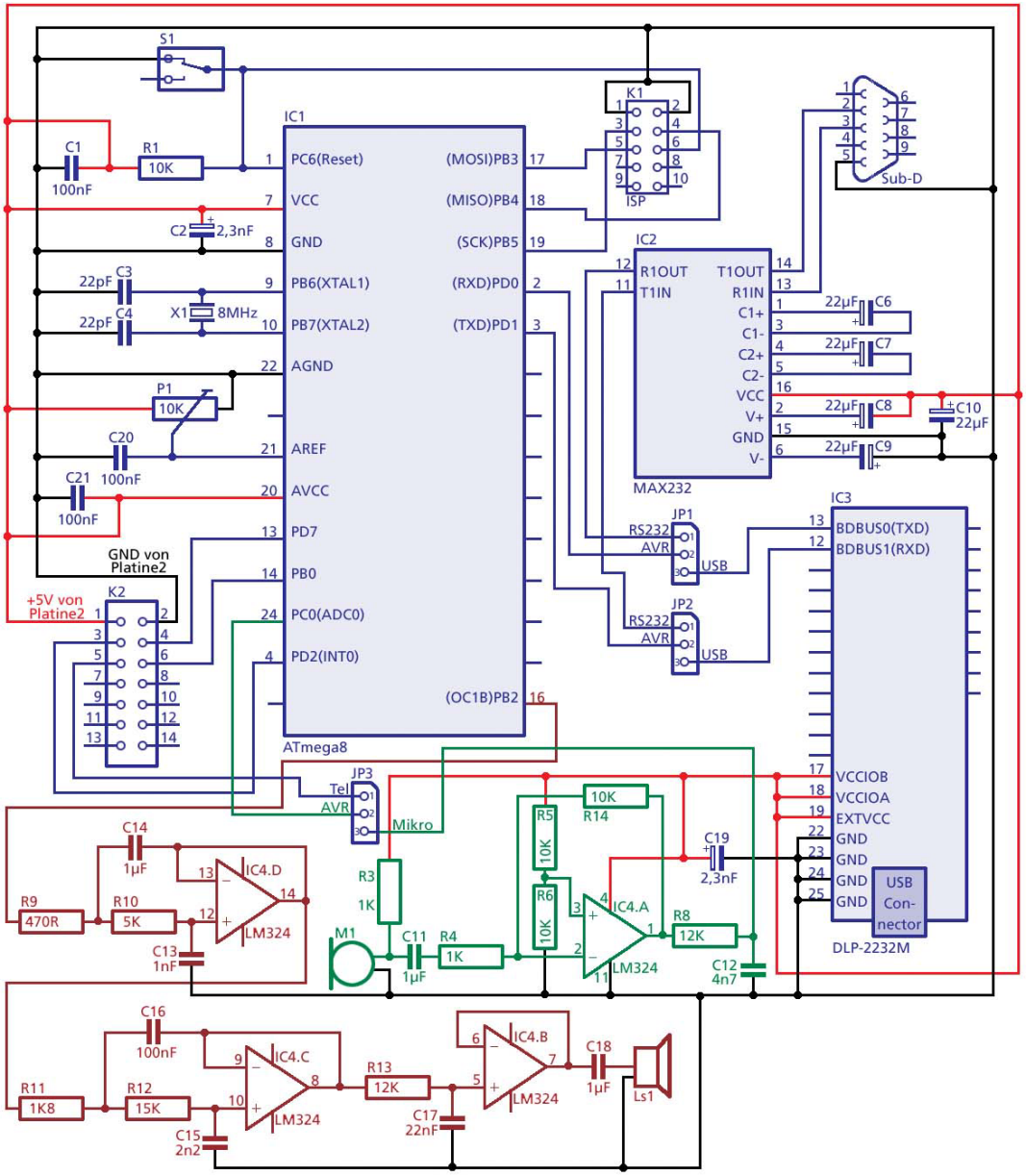


Abbildung 3.2.: Bestückungsplan von Platine 1



Mikrofon-Schaltung
Lautsprecher-Schaltung

Abbildung 3.3.: Schaltplan von Platine 1

Legende:

IC Integrated Circuit (integrierter Schaltkreis)
X *Quarz*
R Widerstand
S *Wechselschalter*
K Steckbuchsen, Steckverbinder
M Mikrofon
Ls Lautsprecher
C Kondensator
T Transistor

3.1.1. ATmega8 anschließen

Für den ATmega8 wird ein 28 poliger IC Sockel verwendet, der auf die Lochrasterplatine aufgelötet wird. In diesen Sockel wird der ATmega8 gesetzt. Dies ermöglicht, dass der ATmega8 bei Bedarf einfach aus der Schaltung herausgenommen werden kann, da er nicht direkt mit der Schaltung verlötet ist. Außerdem wird er dadurch nicht mit der Löthitze in Verbindung gebracht, was gegebenenfalls Schaden am ATmega8 anrichten kann.

3.1.2. Spannungsversorgung

Der ATmega8 benötigt eine stabile Spannungsversorgung von 4,5 bis 5,5 Volt [ATmega8]. Da die Platine 1 von Platine 2 mit 5V über die Steckverbindung K2 versorgt wird, wird das genaue Zustandekommen dieser 5V bei der Beschreibung von Platine 2 erklärt. Jedenfalls werden die 5V, die von Platine 2 kommen, mit den Pins VCC (+5V) und GND (Masse) des ATmega8 verbunden.
Die zwei großen gepolten *Tantal-Elektrolytkondensatoren* C2 und C19 sollen durch ihre Trägheit beim Entladen kurzzeitige Spannungsabfälle in der Schaltung überbrücken.

3.1.3. Reset-Schalter

Verbindet man den Pin RESET (PC6) des ATmega8 mit GND (Masse), dann wird ein Reset des Programms auf dem ATmega8 ausgelöst. Der *Wechselschalter* S1 übernimmt die Aufgabe, bei Umschalten einen Reset am ATmega8 auszulösen.
Der ATmega8 hat zwar einen internen *Pull-Up-Widerstand*, der den Reset-Pin wieder gegen VCC „zieht“, dieser ist jedoch sehr hochohmig und reicht unter Umständen nicht aus. Um den Reset-Pin sicher wieder hochzuschalten empfiehlt es sich einen externen *Pull-Up-Widerstand* R1 zu verwenden, der den Reset-Pin mit VCC verbindet. Zusätzlich sollte man noch einen Kondensator C1 zwischen Reset-Pin und GND anordnen. Dieser Kondensator sorgt dafür, dass der Reset-Eingang gegenüber *Spikes und Glitches* unempfindlich wird. Der Kondensator C1 sollte für diese Aufgabe nah am ATmega8 montiert werden [Mikro].

3.1.4. Externer Taktgeber

Der ATmega8 wird mit einem aktivierten internen Taktgeber von 1 MHz ausgeliefert. Leider ist dieser Taktgeber nicht 100% exakt. Um nun einen exakten Takt zu erhalten und den ATmega8 außerdem schneller zu machen, wird in dieser Arbeit ein externer Quarz (X1) mit 8 MHz genutzt.

Zum Anschwingen des Quarzes werden die zwei Kondensatoren C3 und C4 verwendet. Die zwei Pins XTAL1 und XTAL2 des ATmega8 werden mit den beiden Anschlüssen des Quarzes und jeweils mit einem der beiden Kondensatoren gegen Masse angeschlossen.

Der Quarz sollte, genauso wie die beiden Kondensatoren, möglichst nahe am ATmega8 platziert werden. Längere Leitungen könnten nämlich wie eine Funkantenne fungieren, was in der Regel zu starken Hochfrequenzensignalen führt, die nicht nur diese, sondern auch andere Schaltungen in der Nähe stören könnten [RoboW].

3.1.5. Betriebsspannung für Analog-Digital-Wandler

Die drei Anschlüsse AVCC, AGND und AREF des Atmega8 haben eine besondere Funktion. Der Analog-Digital-Wandler des ATmega8 benötigt eine eigene Versorgungsspannung, die er über diese drei Anschlüsse erhält.

Die Referenzspannung, die am Pin AREF anliegt, kann eine beliebige Spannung zwischen VCC und GND sein. Über das *Potentiometer* P1 ist diese Spannung einstellbar. In dieser Schaltung ist das *Potentiometer* auf 2,6 K Ohm eingestellt, und erzeugt damit eine Referenzspannung VREF von 2,45V

Das analoge Sprachsignal, das dann am Eingang des ATmega8 anliegt, darf nur zwischen 0V und AREF liegen.

3.1.6. Mikrofon-Schaltung

Die Applikation Note AVR335 von Atmel [AVR335] bietet hauptsächlich die Vorlage für die Mikrofon- und Lautsprecher-Schaltung auf Platine 1.

Das analoge Mono-Sprachsignal kommt normalerweise vom Wählscheibentelefon und soll für die Übertragung zum PC vom ATmega8 digitalisiert werden. Um dies zu testen wurden auf Platine 1 zusätzlich ein Anschluss und Filter für ein hier anschließbares Mikrofon (M1)

angebracht. Der Filter ist ein invertierender Verstärker, der die schwachen Sprachsignale vom Mikrofon für den ATmega8 verstärkt.

Im Schaltplan von Platine 1 ist die Mikrofon-Schaltung grün dargestellt.

Durch den Jumper JP3 kann man wählen, ob das analoge Sprachsignal vom Mikrofon der Platine 1 an den Atemga8 geleitet wird, oder vom Wählscheibentelefon, das an Platine 2 angeschlossen ist.
Ohne diesen Jumper würde man beide Signale an jeweils einen analogen Eingang des Atmeag8 anschließen und müsste dann in der Software angeben, welcher analoge Eingang genutzt werden soll. Der Jumper ermöglicht nun ein Umschalten ohne die Software verändern zu müssen.

Der Widerstand R3 wird in der Mikrofon-Schaltung benutzt, um das Mikrofon mit Energie zu versorgen. Der Kondensator C11 blockt hierbei alle Gleichstrom-Komponenten zum Verstärker.

Als *Operationsverstärker* wird der vierfach Universaloperationsverstärker LM324 [LM324] verwendet. Er hat vier *Operationsverstärker* integriert, von denen einer (IC4.A) für die Mikrofon-Schaltung und die anderen drei (IC4.B, IC4.C, IC4.D) für die im nächsten Abschnitt erklärte Lautsprecher-Schaltung verwendet werden.

Die vom Mikrofon kommenden Sprachsignale werden mit dem *Operationsverstärker* IC4.A und durch die Widerstände R14 und verstärkt (Verstärkung V = R14/R4, V=10/1, V = 10, das Signal wird somit um das 10-fache verstärkt).
R5 und R6 ergeben einen Spannungsteiler, der die 5V Versorgungsspannung auf 2,5V halbiert und diese an den nicht invertierenden Eingang (3) des *Operationsverstärker*s (IC4.A) weitergibt. Mit Hilfe des Ausgangssignals (1) versucht der *Operationsverstärker* nun die Differenzspannung zwischen beiden Eingangsspannungen (2 und 3) auf Null zu halten.

Der Widerstand R8 und der Kondensator C12 stellen einen einfachen *Tiefpassfilter* erster Ordnung dar. Dieser Tiefpassfilter lässt alle Frequenzen unterhalb der Grenzfrequenz durch und filtert die Frequenzen, die höher sind heraus. Die Grenzfrequenz kann man mit dem Widerstand und dem Kondensator einstellen. Sie berechnet sich bei diesem Tiefpassfilter wie folgt:
R = 12K, C = 4n7F

$fg \approx 1 / (2 * \pi * R * C)$
$fg \approx 1 / (2 * 3{,}14 * 12000\ \Omega * 0{,}0000000047\ F)$
$fg \approx 2821$
$fg \approx 2{,}8\ kHz$

Somit werden alle Frequenzen über 2,8 kHz durch den Tiefpassfilter herausgefiltert.

Zusätzlich beschützt der Widerstand R8 den Verstärker vor Schäden, falls der Ausgang (1) kurzgeschlossen wird.

Nach dieser Filter-Schaltung, wird das verstärkte analoge Sprachsignal über den Jumper JP3 an den analogen Eingang (ADC0) des ATmega8 weitergeleitet.

3.1.7. Lautsprecher-Schaltung

Das Sprachsignal, das vom Gesprächspartner über das Internet verschickt wird, und somit digital ist, muss für die Ausgabe am Telefonhörer wieder analogisiert werden. Diese Aufgabe übernimmt der ATmega8. Auf Platine 1 ist hierfür ebenfalls ein Filter für die Aufbereitung des Sprachsignals angebracht.

Im Schaltplan von Platine 1 ist die Lautsprecher-Schaltung braun dargestellt.

Da der ATmega8 keinen Digital-Analog-Wandler besitzt, wird diese Funktion durch die *Pulsweitenmodulation* (weiter unten im Text wird dieses Verfahren genauer erklärt) nachgebaut. Das entstandene Signal wird am PWM-Ausgang (OC1B) des ATmega8 ausgegeben. Durch die *Pulsweitenmodulation* hat das Signal unerwünscht hohe Frequenzen, die herausgefiltert werden müssen. Die Filter in der Lautsprecher-Schaltung entfernen die hohen Frequenzen und lässt das geglättete analoge Sprachsignal übrig.

Genauer besteht die Lautsprecher-Schaltung aus zwei gestaffelten und aufeinander abgestimmten Tschebyscheff-Filtern zweiter Ordnung (IC4.D, R9, R10, R11, C13, C14 und IC4.C, R11, R12, R13, C15, C16), und einem Tiefpassfilter erster Ordnung (R13, C17). Die Tschebyscheff-Filter werden hier verwendet, da diese Filter die Eigenschaft haben, Frequenzen, die unterhalb der Grenzfrequenz liegen, bestmöglich durchzulassen und die Frequenzen oberhalb der Grenzfrequenz rabiat herauszufiltern. Bei einem einfachen Tiefpassfilter ist der Bereich um die Grenzfrequenz nicht so gut gelöst. In Abbildung 3.4. ist der Kurvenverlauf der beiden Filter Tschebyscheff und Tiefpass erster Ordnung um die Grenzfrequenz gut erkennbar:

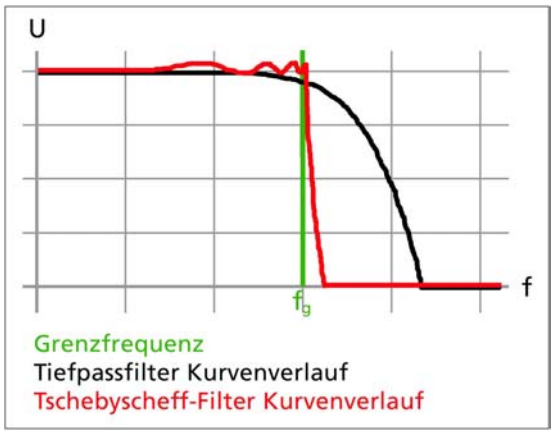


Abbildung 3.4.: Filter-Kurvenverlauf

Der letzte *Operationsverstärker* (IC4.B) in der Lautsprecher-Schaltung beschützt die Schaltung vor Rückkopplung.

Die Grenzfrequenz fg des gesamten Lautsprecher-Filters wurde auf 4 kHz gesetzt, so dass die hohen unerwünschten Frequenzen des Sprachsignals herausgefiltert werden, jedoch das gewünschte Analogsignal nicht verändert wird. Um eine Klangverschlechterung am Lautsprecher (Ls1) durch die Gleichspannungskomponente zu vermeiden, wird der Kondensator C18 verwendet.

3.1.8. SPI-Schnittstelle

Die SPI-Schnittstelle wird unter Anderem benötigt, um den Flash-Speicher des ATmega8 zu programmieren, und das, ohne den ATmega8 aus der Schaltung herausnehmen zu müssen. Ein großer Vorteil der SPI-Schnittstelle ist außerdem ihre Geschwindigkeit. Für das Übertragen von Daten wird ein *ISP*-Programmieradapter benötigt. Dieser wird über einen 10-poligen Wannenstecker (K1) mit Platine 1 verbunden und über den parallelen Druckerport an den PC angeschlossen. Einen Programmieradapter kann man entweder fertig kaufen oder man baut sich selbst einen. Eine Bauanleitung hierfür findet man im Internet z. B. unter folgendem Link:
http://www.roboternetz.de/wissen/index.php/AVR-ISP_Programmierkabel

Abbildung 3.5.: ISP-Programmieradapter

Für die Belegung des 10-poligen Steckers gibt es keinen Standard. Hier wurde der BSD-Programmieradapter (Brian Dean's Programmer, http://www.bsdhome.com/avrdude/) verwendet, der auch als Standard-Einstellung beim Programm AVRDude – das zum Übertragen des Programms zum ATmega8 verwendet wird – eingestellt ist. Die weitere SPI-Schaltung auf Platine 1 muss sich dann nach dem gewählten ISP-Programmieradapter richten.

Der 10-polige Stecker auf Platine 1 wird lediglich mit 4 Pins (MISO, MOSI, SCK und RESET) des ATmega8 verbunden. Mit Pin RESET wird er deswegen verbunden, weil SPI die Reset-Leitung kontrolliert und nach dem Hochladen des Programms auf den Mikrocontroller ein Reset ausgeführt wird. Die in dieser Arbeit gewählte Anschlussbelegung für den 10-poligen Stecker ist aus dem Schaltplan der Platine 1 entnehmbar.

3.1.9. RS232-Schnittstelle

Um Daten und Texte zwischen ATmega8 und PC zu übertragen ist eine RS232-Schnittstelle sehr gut geeignet. Da der ATmega8 über einen internen UART verfügt – ein Modul, das Daten über die RS232-Schnittstelle zum PC senden bzw. auch von ihm empfangen kann – ist eine serielle Übertragung schnell realisierbar.

Da aber die PC-Schnittstelle der Norm entsprechend mit +12V/-12V arbeitet, und der ATmega8 mit 0V/5V, muss unbedingt ein Schaltkreis dazwischen, welcher die Pegel anpasst. Hierfür wird ein MAX232 (IC2) verwendet, der die Spannungsumsetzung realisiert.

Der Pegelwandler MAX232 wird über einen 16-poligen IC Sockel mit Platine 1 verlötet und, wie aus dem Schaltplan von Platine 1 entnehmbar, mit 5 gepolten *Tantal-Elektrolytkondensatoren* (C6 bis C10) verschaltet.

Das für den PC auf +12V/-12V gewandelte Signal wird über einen 9-poligen Sub-D-Stecker, bei dem 3 Pole belegt sind, übertragen. Für die Verbindung zwischen dem Sub-D-Stecker und dem seriellen Port des PC's wird ein 9-poliges Modemkabel verwendet.

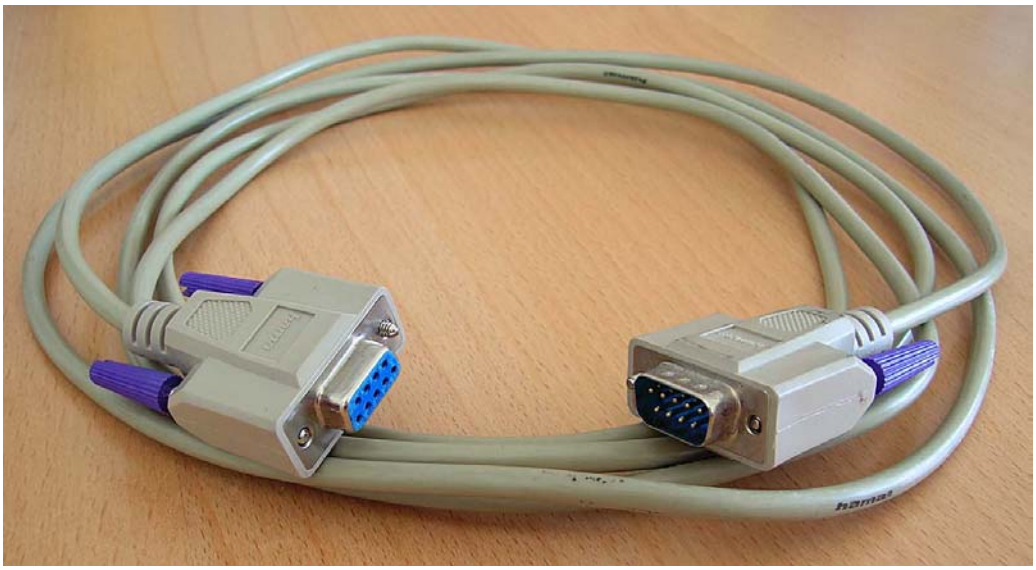


Abbildung 3.6.: Modemkabel mit Sub-D-Stecker

3.1.10. USB-Schnittstelle

Zur Möglichkeit, über die RS232-Schnittstelle zu kommunizieren, kam außerdem die Kommunikationsmöglichkeit über USB dazu. Per USB können Daten schneller übertragen werden. Bei Schaltungen mit geringem Stromverbrauch ist es sogar möglich, die Schaltung über den USB-Bus ohne externe Versorgung mit Strom zu versorgen.

Mit den Jumpern JP1 und JP2 auf Platine 1 kann man zwischen den beiden Übertragungsmöglichkeiten RS232 und USB wählen.

Für die Realisierung der USB-Schnittstelle ist ein Chip der schottischen Firma FTDI ideal. Dieser FTDI-Chip vermittelt zwischen PC und ATmega8 und macht dabei eine USB zu seriell Umsetzung. Dabei ist eine Seite des FTDI-Chips RS232 und die andere USB.

Die Übertragung von Daten erfolgt auf der PC-Seite über die standardmäßigen COM-Port-Funktionen, so dass bei der Erweiterung auf FTDI keinerlei Änderung der Applikation erforderlich ist.

Einstellungen bezüglich der Übertragungsrate werden ignoriert. Ungeachtet der im Programm eingestellten *Baudrate* erfolgt die Übertragung der Daten über USB immer mit der höchstmöglichen Geschwindigkeit. Wird der FTDI-Chip als UART konfiguriert, können serielle Daten mit bis zu 920 kBaud/s über USB übertragen werden.

In dieser Arbeit wird der FTDI-Chip FT2232 verwendet. Dieser kompakte Chip ist ein SMD-Baustein und kann durch seine Bauform nicht ohne Weiteres auf Platine 1 gelötet werden. Aus diesem Grund entschied ich mich für ein fertiges USB-Interfacemodul (IC3) von „DLP-Design", das bereits einen FT2232 beinhaltet und zusätzliche für dessen Betrieb nötige Bauteile wie z. B. ein 4 MHz Quarz. Das Modul mit der genauen Bezeichnung DLP2232M realisiert einen zweikanaligen USB-Umsetzer für serielle und parallele Schnittstellen.

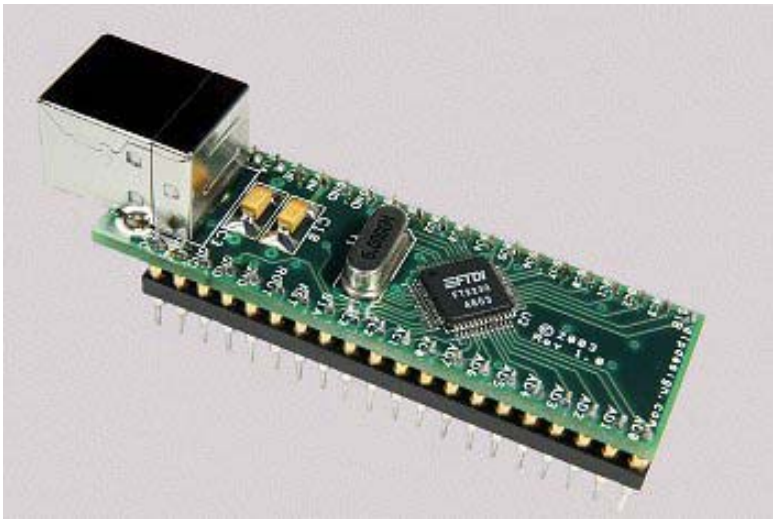


Abbildung 3.7.: USB-Interfacemodul DLP2232M

Das USB-Modul wird auf Platine1 gelötet und erhält von dort auch die 5V Betriebsspannung über die Pins 17, 18, und 19. Die Pins 22, 23, 24 und 25 des USB-Moduls sind mit Masse verbunden. Die Leitungen RxD (Recive Data) und TxD (Transmit Data) des USB-Moduls werden bei der Verbindung mit den RxD und TxD Pins des ATmega8 gekreuzt.

Um vom PC aus mit dem USB-Modul kommunizieren zu können, ist ein Treiber nötig. FTDI Treiber sind kostenlos unter http://www.ftdichip.com/FTDrivers.htm erhältlich.

Für die meisten Betriebssysteme sind hierbei zwei Treibervatianten erhältlich:

- VCP-Treiber (Virtuell COM-Port)
  Der VCP-Treiber emuliert am PC eine normale serielle Schnittstelle (COM). Das USB-Gerät kann damit vom PC als normales RS232-Gerät angesprochen werden.
- D2XX-Treiber
  Mit dem D2XX-Treiber kann man über eine DLL-Schnittstelle direkt auf das USB-Gerät zugreifen.

Das USB-Modul kann viele Arbeitsmodi annehmen und muss daher zunächst für die eigenen Bedürfnissee konfiguriert werden. Dies wird mit dem Programm „MProg“ vorgenommen. Das Programm MProg funktioniert nur mit einem D2XX-Treiber und kopiert diesen bei Installation auf die Festplatte.
Verbindet man nun das USB-Modul per USB mit dem PC, so wird per Plug-and-Play-Funktion nach einem Treiber gefragt. Daraufhin installiert man den D2XX-Treiber des Programms MProg.

Beim Öffnen des Programms erscheint unten abgebildetes Fenster. Im Bearbeitungs-Modus kann man dann eine neue Programmier-Vorlage erstellen, die auch speicherbar ist.

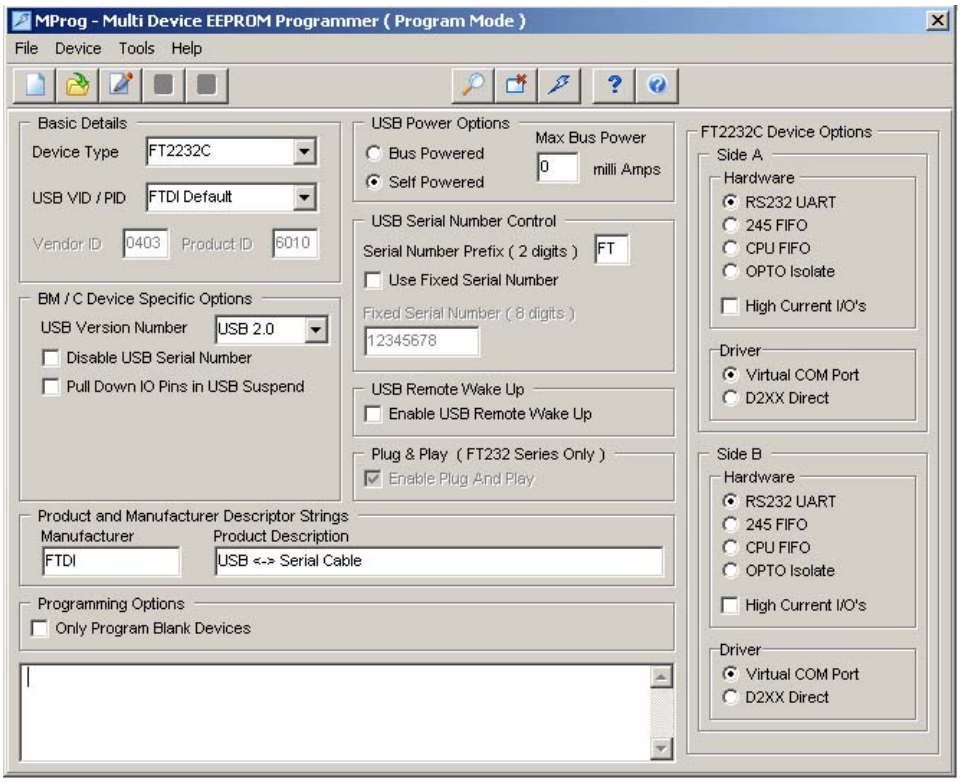


Abbildung 3.8.: Das Programm MProg

Ist man nun im Bearbeitungs-Modus des Programms, kann man alle Optionen im Fenster ändern. Links oben wählt man z.B. den verwendeten FTDI-Chip-Typ aus, und die IDs des Chips, hier werden die Standard-IDs verwendet.

In der zweiten Spalte des Fensters gibt man oben an, ob der FTDI-Chip eine externe Betriebsspannung erhält oder ob er sie über den USB-Bus vom PC erhalten soll.

Ganz rechts wird dem EEprom mitgeteilt, welcher Treiber für die Kommunikation zwischen FTDI-Chip und PC verwendet werden soll und welche Übertragungsart gewünscht ist. Weil der FT2232C zwei USB-Umsetzer integriert hat, kann man diese Einstellungen für zwei Kanäle, A und B, angeben.

Alle anderen Einstellungen des MProg-Fensters werden so beibehalten.

Hat man die Einstellungen gespeichert, so kann man im Programmier-Modus des Programms MProg diese Einstellungen in das EEprom des USB-Moduls programmieren.

Für die Kommunikation wurde im weiteren Betrieb der VCP-Treiber gewählt, der nach dem Programmieren des EEproms auf dem PC installiert werden muss und somit den D2XX-Treiber ersetzt. Der VCP-Treiber bietet den Vorteil, dass er dem PC zwei virtuelle COM-Ports erzeugt.

3.1.11. Verbindung mit Platine 2

Platine 1 ist mit Platine 2 über einen 14-poligen Wannenstecker (K2) verbunden. Ein 10-poliger Stecker hätte auch ausgereicht, ich habe mich aber trotzdem für einen 14-poligen entschieden, um ein Falschanschließen mit dem 10-poligen Stecker der SPI-Schnittstelle auszuschließen.

3.2. Lochrasterplatine 2

Die zweite Lochrasterplatine wird mit einer 9V Gleichspannungsquelle versorgt. Aus diesen 9V bereitet ein Spannungsregler die 5V, die an Platine 1 weitergereicht werden. Die Signale des Telefons kommen an Platine 2 an und werden hier für den ATmega8 aufbereitet. Aus dem Telefonsignal wird mit Hilfe verschiedener Teilschaltungen das Wählsignal, der Telefonhörerstatus und das Sprachsignal aufgedröselt. Zusätzlich gibt es für Platine 2 noch eine 40V Wechselspannungsquelle, die für das Betätigen der Telefonklingel eingesetzt

wird. Der Einsatz eines *Relais* mit zwei Umschaltern auf Platine 2 ermöglicht eine Einspeisung der 40V für das Telefonklingeln ohne die restliche Schaltung zu belasten.

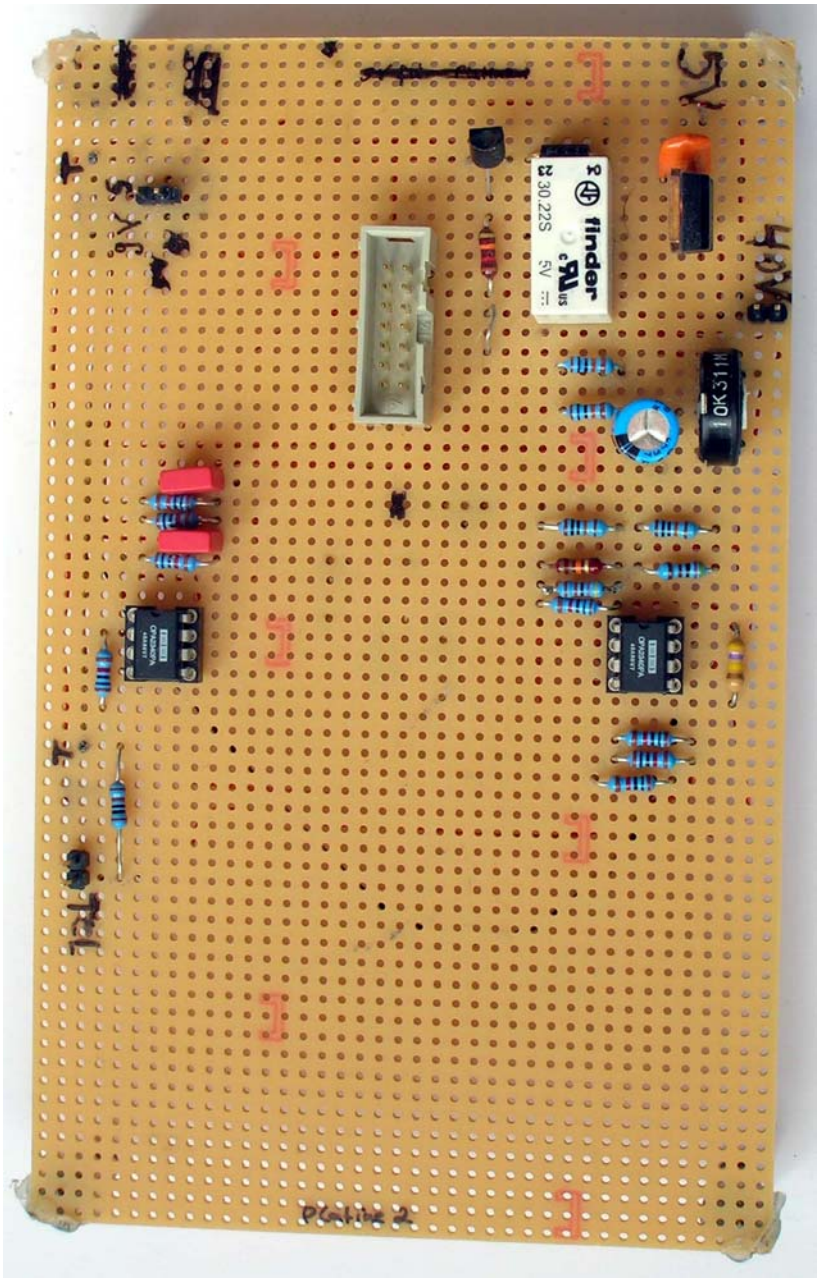


Abbildung 3.9.: Lochrasterplatine 2

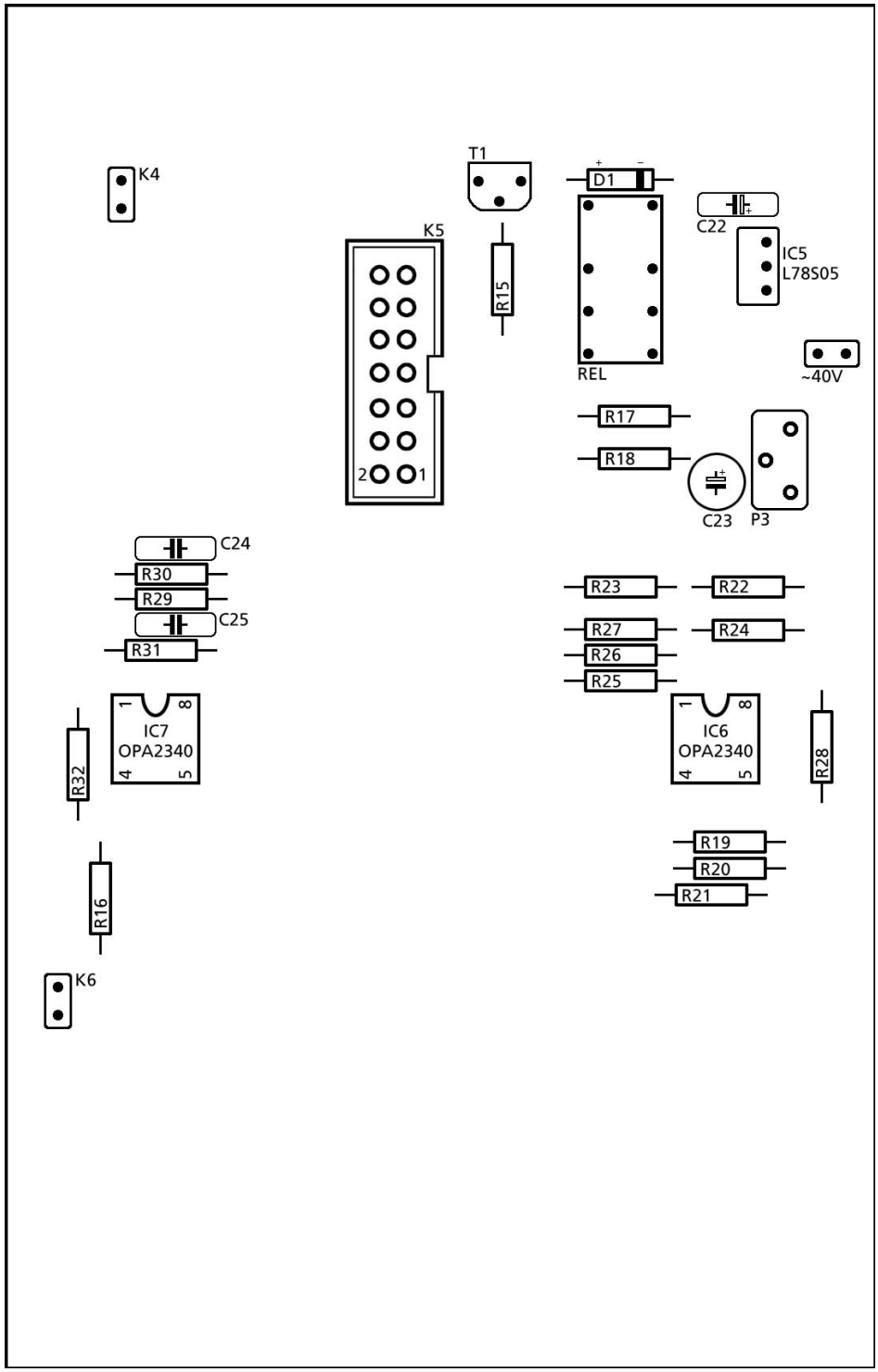


Abbildung 3.10.: Bestückungsplan von Platine 2

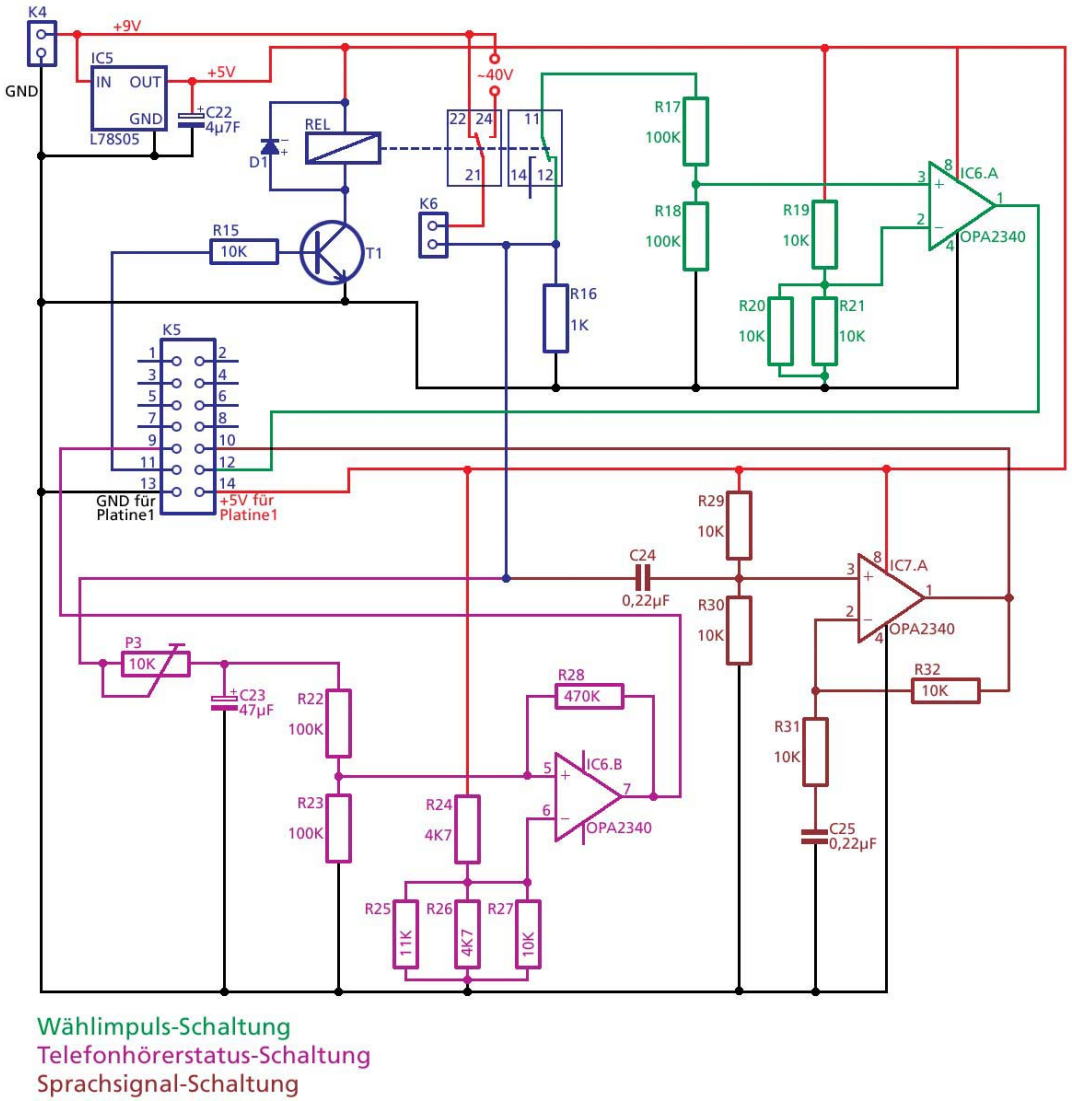


Abbildung 3.11.: Schaltplan von Platine 2

3.2.1. 9V Gleichspannung

Platine 1 und 2 werden grundsätzlich mit der 9V-Gleichspannung versorgt. Zum einen benötigt das Telefon 9V, mit dessen 9V-Signalen auch die gesamte Schaltung auf Platine 2 weiterarbeitet, zum anderen werden die 9V von einem Spannungsregler (IC5) in 5V umgewandelt, die Platine 1 über den Steckverbinder K5 zugeführt werden.

3.2.2. Spannungsregler

Der Spannungsregler (IC5) wandelt eine höhere Eingangsspannung – in diesem Fall 9 V – immer genau in 5V um. Dies kommt auch Platine 1 zugute, die diese stabilen 5V über die Steckverbindung K5 (13, 14) von Platine 2 erhält.
Als Spannungsregler wird hier der Typ L78S05CV [L78S05] verwendet, welcher 2A verträgt, sowie über einen Kurzschluss- und Überlastungsschutz verfügt.
Der gepolte Elektrolytkondensator C22 ist wichtig, da er die Spannung, die vom 9V Netzgerät kommt, glättet.

3.2.3. Schutzwiderstand

Der Widerstand R16 ist ein Schutzwiderstand für eventuelle Kurzschlüsse.

3.2.4. 5V-Relais

Das *Relais* (REL) mit 2 Umschaltkontakten benötigt 5V, um beide Schalter umzuschalten. Sobald diese 5V nachlassen, gehen die Schalter wieder in ihre Ausgangsposition zurück.
Die Diode D1 arbeitet als Schutzdiode, um die beim Schalten des Relais entstehenden Induktionsspannungen abzuleiten.
Der *NPN-Transistor* T1 verhindert, dass das Relais beim Anlegen der Betriebsspannung oder bei kurzen Spannungsunterbrechungen ungewollt schaltet.
Die 5V zum Umschalten des Relais kommen vom ATmega8 über den Steckverbinder K5 (11). Schickt der ATmega8 darüber nun diese 5V, so schalten die beiden *Wechselschalter* gleichzeitig um. Einer der Wechselschalter bewirkt durch das Umschalten, dass der Telefonschaltung eine 40V-Wechselspannung zugeführt wird und somit das Telefon klingelt. Der andere Wechselschalter öffnet durch das Umschalten einen Stromkreis, damit dieser Stromkreis von den 40V verschont bleibt.
Wird nun der Telefonhörer abgehoben, ändert sich das Telefonsignal. Diese Änderung muss an den ATmega8 weitergegeben werden, der wiederum die 5V für das Relais abschaltet. Die beiden Schalter des Relais gehen in ihre Ausgangsposition und die 40V verschwinden aus der Schaltung.

3.2.5. 40V Wechselspannung

Um die Telefonklingel zu betätigen, werden kurzzeitig 40V Wechselspannung auf die 9V Gleichspannung gelegt. Wie bereits beschrieben, werden die 40V durch das Schalten der Relais-Schalter der Telefonschaltung zugeführt.

3.2.6. Telefonanschluss

Das Telefon wird über den Steckverbinder K6 mit Platine 1 verbunden. Es wird konstant mit der 9V Gleichspannungsquelle versorgt. Soll das Telefon klingeln, schaltet das Relais und es werden 40V Wechselspannung zusätzlich in den Telefon-Stromkreis gespeist.

3.2.7. Telefonsignalverarbeitung

Wie in Kapitel 2.1. erwähnt, hat das Telefon eine Leitung, die alle für das Telefonieren relevanten Signale enthält. Ein Grossteil der Schaltung auf Platine 2 dient dazu, aus dieser Leitung die Wählsignale, den Telefonhörerstatus und die Sprachsignale heraus zu filtern. Die einzelnen Zustände sind hierbei eindeutig auseinanderzuhalten.

3.2.8. *Operationsverstärker*

Um die Telefonsignale für den Atmeag8 aufzubereiten, werden auf Platine 2 zwei *Operationsverstärker* mit der genauen Bezeichnung OPA2340PA [OPA2340] (IC6, IC7) verwendet. Mit Hilfe von zwei 8-poligen IC Sockeln werden diese auf die Platine gelötet. Jeder OPA2340 hat zwei *Operationsverstärker* integriert, die im Schaltplan durch A und B (IC6.A, IC6.B) gekennzeichnet sind. Als Betriebsspannung benötigen die OPA2340s 5V. Im Folgenden werden die einzelnen Einsatzgebiete erklärt.

3.2.9. Wählimpuls-Schaltung

Die Wählimpuls-Schaltung ist im Schaltplan der Platine 2 grün dargestellt.

Mit dieser Schaltung werden im Speziellen die Wählsignale des Telefons für den ATmega8 aufbereitet.

Die eventuelle 40V Wechselspannung ist in der Wählimpuls-Schaltung störend, und außerdem überflüssig, weil man – wenn das Telefon klingelt – gewöhnlich keine Ziffer wählt. Aus diesem Grund hat das Relais (REL) zwei *Wechselschalter*. Mit dem ersten *Wechselschalter* wird die 40V Wechselspannung eingeschaltet, der zweite *Wechselschalter* schaltet immer gleichzeitig mit dem ersten und bewirkt in dem Moment eine Unterbrechung des Wählimpuls-Stromkreises.

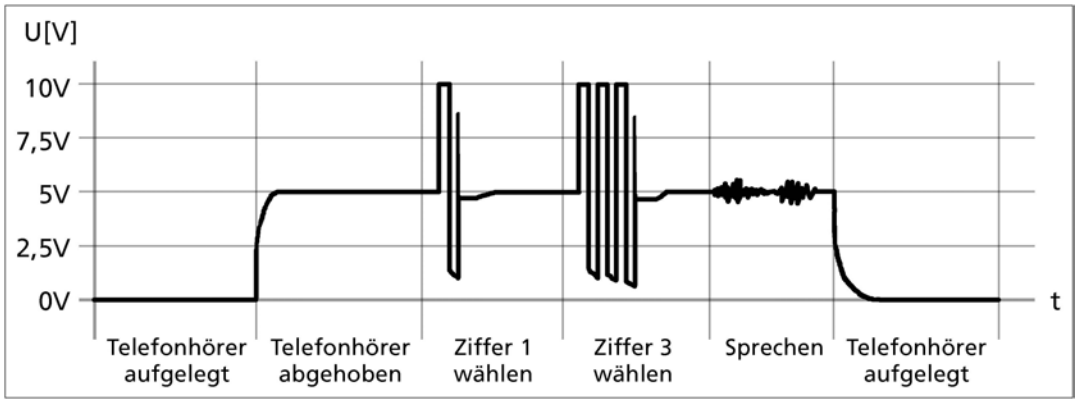


Abbildung 3.12.: Ursprungsform Telefonsignal

In Abbildung 3.12. ist das Telefonsignal in seiner Ursprungsform dargestellt. Dieses Telefonsignal, das sich zwischen 0V und 10V bewegt, gilt es nun für den ATmega8 aufzubereiten, d. h. das Signal soll sich zwischen 0V und 5V bewegen und die Wählimpulse eindeutig anzeigen.

Das Telefonsignal wird nach dem Relais (REL) zunächst durch den Spannungsteiler, bestehend aus R17 und R18, halbiert. Das Signal bewegt sich nun also nur noch zwischen 0V und 5V. Dieses Signal wird an den nichtinvertierenden Eingang (3) des *Operationsverstärkers* IC6.A weitergeleitet. Durch die Widerstände R19, R20 und R21, lässt sich für den invertierenden Eingang (2) des Operationsverstärkers IC6.A eine konstante Spannung von 1,75V erzeugen. Diese 1,75V werden auch „Schwelle“ genannt, denn: Ist die Spannung am Eingang 3 kleiner als die Spannung am Eingang 2, so liegt am Ausgang (1) des *Operationsverstärkers* die volle negative Betriebsspannung des *Operationsverstärkers* – hier 0V – an. Ist die Spannung am Eingang 3 größer als die Spannung am Eingang 2, so liegt am Ausgang (1) dagegen die volle positive Betriebsspannung (5V) an. Die Schwelle 1,75V ist daher ausschlaggebend für das entstehende Signal, und bewirkt am Ausgang (1) des *Operationsverstärkers eine* eindeutige Signalumschaltung ohne Verzögerung.

Da der *Operationsverstärker* außerdem keine weitere Beschaltung hat, arbeitet er hier als Komparator. Wie der Name sagt, vergleicht er ein Eingangssignal mit einem Vergleichssignal.

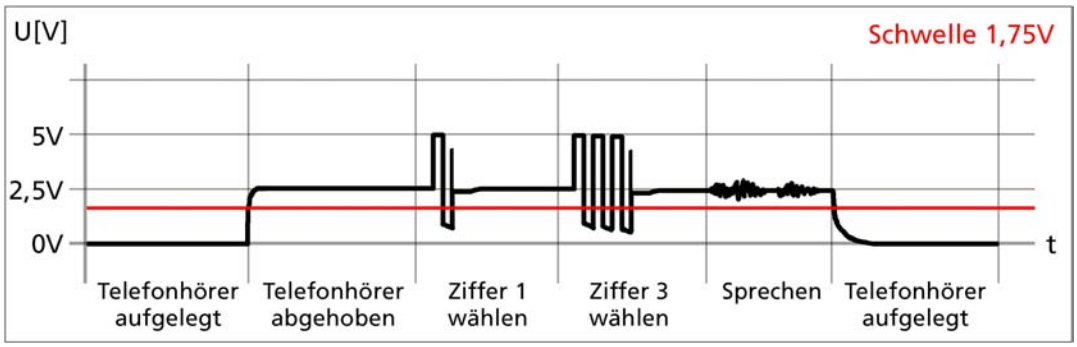


Abbildung 3.13.: Signale an den Eingängen 3 (schwarz) und 2 (rot)

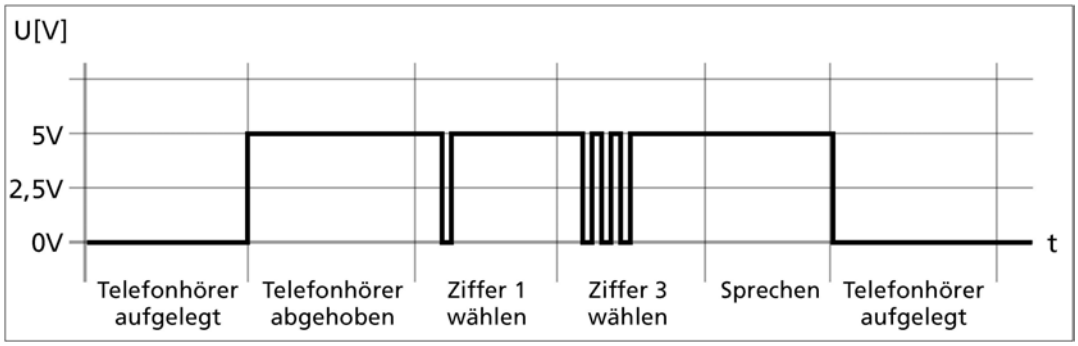


Abbildung 3.14.: reines Signal nach Wählimpuls-Schaltung

In Abbildung 3.14. ist das Signal nach dem *Operationsverstärker* IC6.A dargestellt. Dieses Signal wird über den Steckverbinder K5 von Platine 2 zu Platine 1 übertragen und dort an den Pin PD2 des ATmega8 weitergeleitet.

3.2.10. Telefonhörerstatus-Schaltung

Die Telefonhörerstatus-Schaltung ist im Schaltplan der Platine 2 lila dargestellt.

Diese Schaltung ist dazu da, um dem ATmega8 mit einem eindeutigen Signal mitzuteilen, ob der Telefonhörer aufgelegt oder abgehoben ist.

Bei der Telefonhörerstatus-Schaltung darf die 40V Wechselspannung nicht wie bei der Wählimpuls-Schaltung mit dem Relais abgeschottet werden, da der Telefonstatus auch während dem Telefonklingeln benötigt wird. Während des Telefonklingelns wird irgendwann der Telefonhörer abgehoben. Dieser Status muss dann an den ATmega8

weitergegeben werden, damit dieser daraufhin die 5V für das Relais abschaltet, welches wiederum die 40V Wechselspannung abschaltet.

Die eventuelle 40V Wechselspannung wird gleich zu Anfang der Telefonhörerstatus-Schaltung durch einen Tiefpassfilter erster Ordnung, der aus dem *Potentiometer* P3 und dem Kondensator C23 besteht, herausgefiltert. Das *Potentiometer* ist dabei auf 2,3 K Ohm eingestellt.

Das Signal nach dem Tiefpassfilter liegt dann – ob nun das Telefon klingelt oder nicht – bei 0V. Wird der Telefonhörer abgehoben, steigt das Signal mit starker Verzögerung auf 5V an. Diese Verzögerung ist auf den Konsenstor C23 zurückzuführen, der sich erst voll aufladen muss. Der Tiefpassfilter bewirkt außerdem, dass sich die in dieser Schaltung unerwünschten Wählsignale zwischen 5V und 10V bewegen. Somit ist es sinnvoll, für den Telefonhörerstatus nur noch das Signal unter 5V zu betrachten. Insgesamt wird das Signal durch den Tiefpassfilter noch träger und somit für den ATmega8 für die Weiterverarbeitung noch schlechter. Mit einem *Operationsverstärker* kann das Signal wieder verbessert werden.

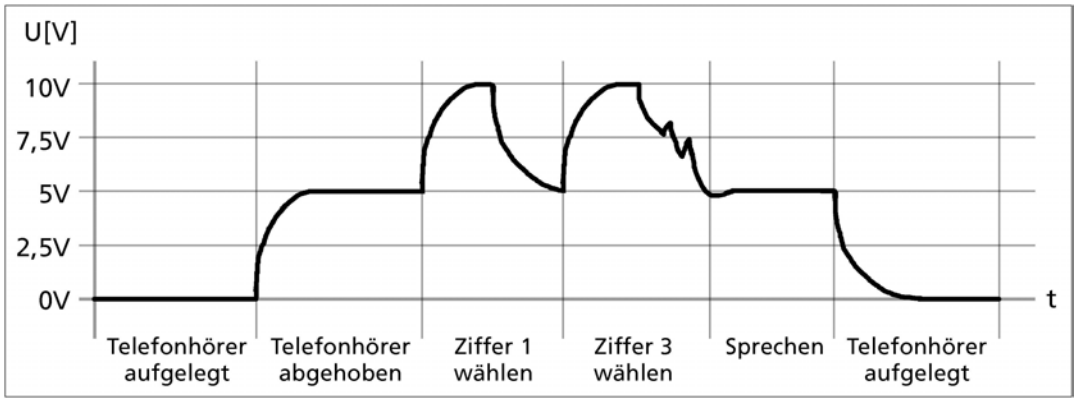


Abbildung 3.15.: Signal nach Tiefpassfilter

Um die Trägheit des Signals zu entfernen und ein sauberes Signal für den ATmega8 zu erzeugen, wird das Signal nach dem Tiefpassfilter zunächst mit einem Spannungsteiler (R22, R23) halbiert (das Signal bewegt sich dann nur noch zwischen 0V und 5V) und an den nichtinvertierenden Eingang (5) des *Operationsverstärkers* IC6.B weitergeleitet. Durch die Widerstände R24 bis R27 wird eine konstante Spannung von 1,75V an den invertierenden Eingang (6) gelegt (Schwelle). Der Widerstand R28 ist eigentlich nur dann nötig, wenn das Signal am Eingang 5 häufig in der nähe der Schwelle (1,75V) liegt. Dies ist jedoch nicht der Fall. Auch in dieser Schaltung wird der *Operationsverstärker* als Komparator verwendet, da er ebenfalls zwei Eingangssignale vergleicht.

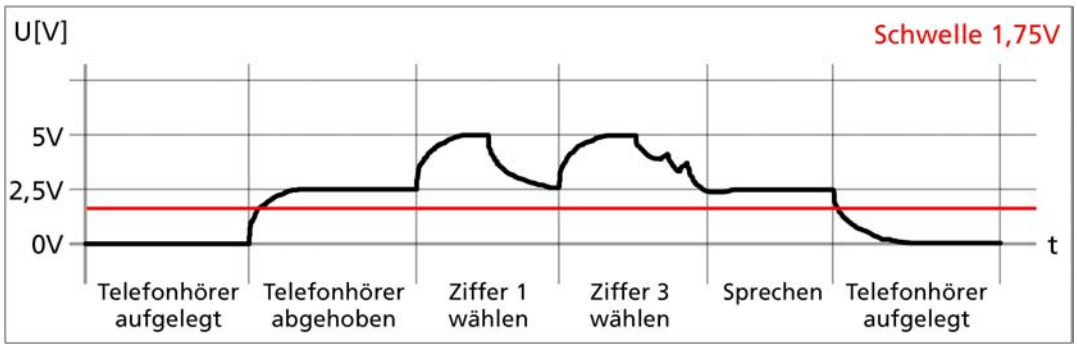


Abbildung 3.16.: Signale an den Eingängen 5 (schwarz) und 6 (rot)

Wie bei der Wählimpuls-Schaltung ist die Schwelle für das Signal am Ausgang (7) des *Operationsverstärkers* ausschlaggebend. Befindet sich das Signal am Eingang 5 unterhalb der Schwelle, dann hat das Signal am Ausgang (7) des *Operationsverstärkers* 0V. Wenn das Signal oberhalb der Schwelle liegt, dann hat das Signal am Ausgang 5V. Somit ergibt sich ein reines Signal, das für den ATmega8 ideal für die Weiterverarbeitung ist.

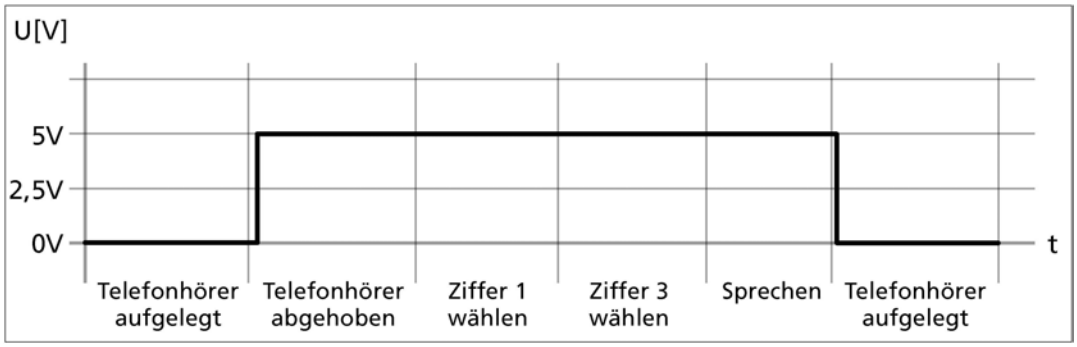


Abbildung 3.17.: Ausgangssignal des *Operationsverstärkers*

Das entstandene Signal wird über den Steckverbinder K5 (9) von Platine 2 zu Platine 1 übertragen und dort an den Pin PB0 des ATmega8 weitergeleitet.

3.2.11. Sprachsignal-Schaltung

Die Sprachsignal-Schaltung ist im Schaltplan der Platine 2 braun dargestellt.

Die analogen Sprachsignale vom Telefonhörer werden mit Hilfe der Sprachsignal-Schaltung um das Zweifache verstärkt.

Die Sprachsignal-Schaltung kann man entweder ebenfalls mit dem Relais vor der 40V Wechselspannung schützen, oder man baut stattdessen einen ungepolten Kondensator C24 in die Schaltung ein, der die danach kommende Schaltung schützt. Wenn das Telefon nicht klingelt, dann hat das Signal vor dem Kondensator C24 – je nachdem ob der Telefonhörer aufliegt oder nicht – 0V bis 5V. Das Sprachsignal befindet sich im Bereich zwischen 500mV und 1V und liegt im Bereich um die 5V.

Der Spannungsteiler, bestehend aus den Widerständen R29 und R30, halbiert die 5V, die vom Spannungsregler kommen, und addiert die so entstandenen 2,5V mit dem Sprachsignal. Das so entstandene Signal wird an den nichtinvertierenden Eingang (3) des *Operationsverstärkers* IC7.A weitergegeben.

Der *Operationsverstärker* wird hier als nichtinvertierender Verstärker (*Schmitt-Trigger*) verwendet, da er eine Gegenkopplung hat, um eine definierte Verstärkungen zu erhalten. Die Verstärkung wird durch das Gegenkopplungsnetzwerk R32 und R31 definiert. Mit Hilfe des Signals am Ausgang (1) versucht er die Differenzspannung zwischen seinen zwei Eingängen auf Null zu halten. Die Ausgangsspannung wird auf den invertierenden Eingang (2) zurückgeführt.

Die Verstärkung erhält man nach folgender Gleichung:
Verstärkung = 1 + R32 / R31
Verstärkung = 2 (das Signal wird verdoppelt)

Das für den ATmega8 verstärkte Sprachsignal befindet sich dann immer um die 2,5V und wird über den Steckverbinder K5 (12) von Platine 2 zu Platine 1 übertragen und dort an den Pin PC0 des ATmega8 weitergeleitet.

3.3. Verbindung der zwei Lochrasterplatinen

In der folgenden Abbildung sind beide Lochrasterplatinen dargestellt. Sie sind miteinander verbunden und alle möglichen Anschlüsse sind belegt.

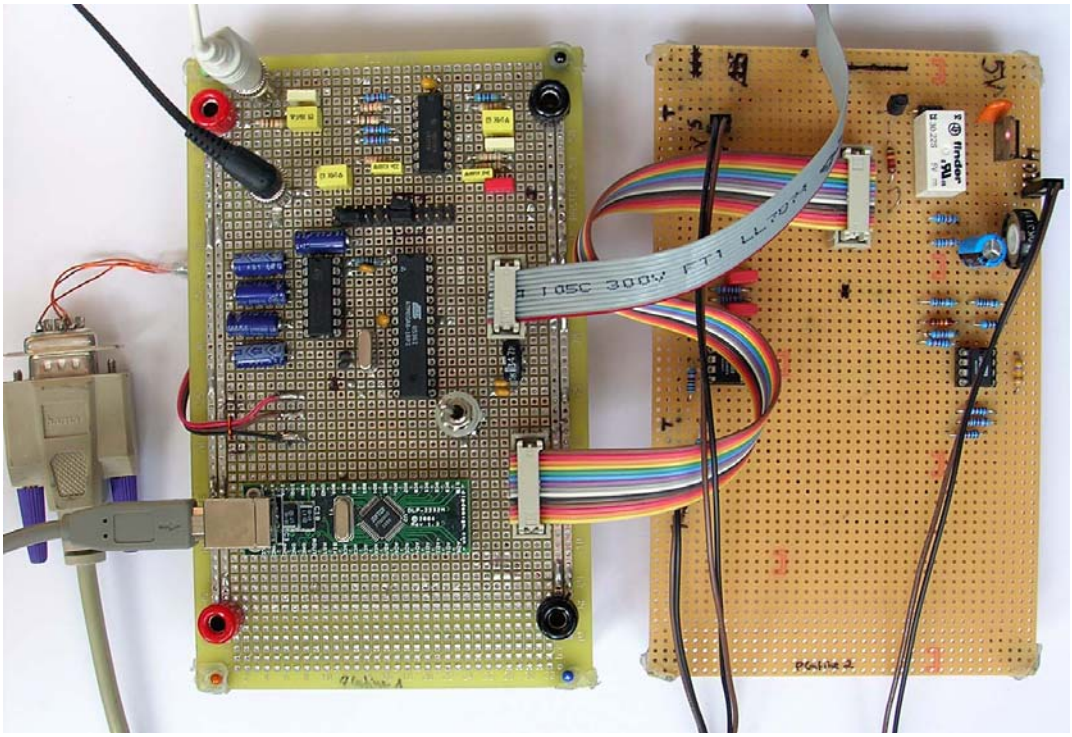


Abbildung 3.18.: Beide Lochrasterplatinen sind miteinander verbunden

4. Verwendete Programme

Für diese Arbeit wurde ausschließlich Open-Source-Software verwendet. Dieses Kapitel beschreibt alle Programme, wobei das wichtigste „WinAVR“ ist, welches als erstes erläutert wird.

4.1. WinAVR

WinAVR ist eine große Sammlung von Open-Source-Tools für die Atmel AVR Mikrocontroller Serie. Das Programm enthält unter Anderem den AVR-GCC Compiler für C und C++, die C-Standardbibliothek AVR-Libc (inkl. Dokumentation), die Programmiersoftware „AVRDude“ und den Editor „Programmer's-Notepad“.

Der schnelle und komfortable Editor „Programmer’s-Notepad“ diente mir als Entwicklungsumgebung. Er bietet sehr nützliche Funktionen wie z. B. Syntax-Highlighting, eine dateiübergreifende Suche und die Möglichkeit externe Tools einzubinden.

Die für diese Arbeit verwendete WinAVR Version ist vom 20.07.2004 (WinAVR-20040720-install.exe)

Die wichtigsten Tools in dieser WinAVR Version haben folgende Versionen:
- AVR-GCC 3.4.1
- AVR-Libc 1.0.4
- AVRDude 4.4.0
- Programmer’s Notepad 2.0.5.32

WinAVR kann man kostenlos unter http://winavr.sourceforge.net/ herunterladen.

4.1.1. AVRDude 4.4.0

Das in WinAVR enthaltene Programm AVRDude ist ein Kommandozeilen-Programm zum up- und downloaden des Flash-Speichers und EEpoms des ATmega8. Außerdem können damit Fuse- und Lock-Bits programmiert werden. Es benutzt hierzu die SPI-Schnittstelle.

Das Programm AVRDude befindet sich nach der Installation des Programms WinAVR im Verzeichnis WinAVR\bin\avrdude.exe.

AVRDude ist, wie bereits erwähnt, in WinAVR enthalten, man kann es aber auch unter http://savannah.nongnu.org/download/avr/avrdude-5.0-w32.zip separat herunterladen.

Die in dieser Arbeit entstandene Software wird mittels AVRDude über einen BSD-Programmieradapter an der parallelen Schnittstelle LPT1 des PCs übertragen.

Bei Eingabe der folgenden Zeile in die Bash bzw. Eingabeaufforderung, wird die Übertragung des Maschinencodes (Hex-Datei) auf den ATmega8 angestoßen.

```
avrdude -p m8 -c bsd -e -U flash:w:cmdlinetest.hex
```

`avrdude` Aufrufen des Programms AVRDude
`-p m8` Angabe des Mikrocontroller-Typ's (ATmega8 → m8)
`-c bsd` Angabe des Programmiergerätes, hier BSD
`-e` Verursacht ein Löschen des Atmega8 vor dem Neuprogrammieren
`-U flash:w:cmdlinetest.hex`
`-U` bewirkt eine Aktion auf einen Speicher des ATmega8
in diesem Fall wird der `flash`-Speicher ausgewählt
`w` gibt an, dass der Speicher beschrieben werden soll
und zwar mit der Datei `cmdlinetest.hex`

4.1.2. Die Datei install_giveio.bat

Für die Ansteuerung von Parallelport-Adaptern unter Windows XP wird ein spezieller Porttreiber (give-io) mitgeliefert. Der Porttreiber kann durch das Ausführen der Datei install_giveio.bat, die beim Installieren von WinAVR in das Verzeichnis WinAVR\bin\install_giveio.bat kopiert wurde, installiert werden.

4.3. Cygwin

Es gibt verschiedene Programme, mit denen das Kompilieren von C-Quelltexten angestossen werden kann. Es ist z. B. mit dem bereits erwähnten „Programmer's Notepad" möglich. Unkomplizierter fand ich jedoch das Kompilieren der C-Dateien mit Hilfe der Shell bash.exe aus dem Cygwin-Paket, das eine UNIX-Umgebung für Windows bietet. An dieser Stelle ist aber auch die Eingabeaufforderung cmd.exe von Windows ausreichend.

Cygwin kann man unter: http://www.cygwin.com/mirrors.html herunterladen.

4.4. HyperTerminal

HyperTerminal erlaubt die Kommunikation zwischen Atmega8 und PC über eine serielle Schnittstelle. Es ist standardmäßig auf jedem Windows-PC installiert und auffindbar unter: Start → Programme → Zubehör → Kommunikation → HyperTerminal

Das Programm muss für eine Verwendung erst Konfiguriert werden. Nach dem Start fragt das Programm zunächst nach einem Namen für die neue Verbindung. Im nächsten Fenster wählt man bei „Verbindung herstellen über:“ die COM-Schnittstelle des PCs aus, an der der ATmega8 angeschlossen ist. Nach dem Drücken auf „OK“ öffnet sich ein neues Fenster, in dem man weitere Einstellungen für die Verbindung vornimmt:

- Bits pro Sekunde: 9600
- Datenbits: 8
- Parität: keine
- Stoppbits: 1
- Flusssteuerung: Kein

Nach Bestätigung dieser Einstellungen, ist eine Verbindung aufgebaut.

Eine Kommunikation über USB ist damit ebenfalls möglich. Der VCP-Treiber des USB-Moduls bietet, wie weiter oben bereits erwähnt, virtuelle COM-Ports an, die mit Hyper-Terminal ausgewählt und verwendet werden können.

4.5. MProg

MProg, in der Version 2.3, wird verwendet, um den EEprom-Speicher des USB-Moduls über USB zu programmieren.

Das Programm kann unter http://www.ftdichip.com/Resources/Utilities.htm#MProg kostenlos heruntergeladen werden.

5. Software

Im Folgenden wird die Entwicklung der Software, die auf den ATmega8 geladen wird, beschrieben. Diese Software hat die Funktion, alle Signale des Telefons zu verstehen und teilweise an den PC weiterzuleiten. Andererseits soll sie Anweisungen des PCs verarbeiten können und dadurch eventuelle Aktionen an die Telefonschaltung weiterleiten.

Die Informationen über die verwendeten Register des ATmega8 habe ich aus dem Datenblatt des ATmega8 entnommen. [ATmega8]

5.1. Programmiersprache

Früher wurde die Software für Mikrocontroller ausschließlich mit der Maschinensprache Assembler erstellt. Da ein Programm in Assembler aber für einen bestimmten Mikrocontroller-Typ geschrieben wird und nicht ohne weiteres für ein anderes Mikrocontroller-Modell verwendet werden kann, habe ich mich dafür entschieden, die Software in C zu schreiben. Mit C kann man Programme erstellen, die für eine ganze Mikrocontroller-Familie verwendbar sind.
Für die Mehrzweck-Programmiersprache C gibt es außerdem viele Bibliotheken die speziell für die AVR-Familie angeboten werden. Eine Software kann in C zudem schneller geschrieben werden und ist einfacher nachzuvollziehen und zu warten.

5.2. Compiler

Der AVR-GCC ist ein kostenloser C-Compiler, mit dem man C-Programme zu ausführbaren Programmen übersetzen kann, die auf Mikrocontrollern der AVR-Familie lauffähig sind. An Sprachen versteht AVR-GCC sowohl C als auch C++.

5.3. Entwicklungsumgebung

Wie bereits in Kapitel 4 beschrieben, gibt es für Windows das Softwarepaket „WinAVR“, das eine AVR-GCC Entwicklungsumgebung bietet. Der mitgelieferte „Programmer's-Notepad“ ist für die Programmerstellung sehr gut geeignet und wurde auch von mir zum Entwickeln der Software verwendet.

5.4. Bibliotheken

5.4.1. AVR-Libc

Die AVR-Libc ist die gebräuchlichste Laufzeitbibliothek zum AVR-GCC C-Compiler, welche den Zugriff auf die Mikrocontroller-Hardware erheblich erleichtert. Die AVR-Libc ist kostenlos erhältlich und für nahezu alle Plattformen und Betriebssysteme geeignet.

Ein ausführliches AVR-Libc Benutzerhandbuch wird unter http://www.linuxfocus.org/common/src2/article352/avr-libc-user-manual-1.0.4.pdf angeboten, das alle in C verfügbaren Funktionen dokumentiert.

Bei Installation des Programms WinAVR wird die AVR-Libc auf den PC kopiert und ist im Verzeichnis WinAVR\AVR\include\avr verfügbar.

Die AVR-Libc ist außerdem unter: http://savannah.nongnu.org/projects/avr-libc/ erhältlich.

Die Bibliothek wird verwendet, indem man die nötigen AVR-Libc-Module wie folgt in den eignen Quelltext einbindet:

```
#include <avr/io.h>       // include I/O definitions (port names, etc)
#include <avr/signal.h>   // include "signal" names (interrupt names)
#include <avr/interrupt.h>// include interrupt support
#include <avr/wdt.h>      // enables watchdog
```

cmdlinetest.c, Zeilen 18 - 22

5.4.2. Procyon AVRlib

Die Procyon AVRlib ist eine weitere kostenlose C-Bibliothek für Mikrocontroller, die aber nicht wie die AVR-Libc grundsätzliche Funktionen für den Zugriff auf die Mikrocontroller-Hardware, sondern höhere unterstützende Funktionen bietet, die dem Zweck dienen, einem Entwickler den Weg zum gewünschten Ziel zu erleichtern.

Procyon bietet zusätzlich fertige Projekte an, die die Procyon AVRlib nutzen. So wie z. B. das Projekt „cmdlinetest", das ein Kommandozeilen-Interface darstellt. Die Datei cmdlinetest.c ist hierbei eine ausführbare C-Datei, die eine mögliche Verwendung der Comandline-Funktionen der Procyon AVRlib aufzeigt.
Kurz beschrieben, setzt dieses Programm einige Pins des Mikrocontrollers als Ein- oder Ausgang und versendet darüber per UART Texte an einen PC, der diese an einem Terminal-Fenster ausgeben kann. Die Funktionen für die UART-Funktionalität sind ebenfalls bereits in einer Datei der Procyon AVRlib enthalten.

Für diese Arbeit wurde das Projekt **cmdlinetest** als Vorlage verwendet und von mir auf meine Bedürfnisse angepasst und mit entsprechenden Funktionen erweitert.

Mit Hilfe des Projekts **cmdlinetest** war somit schon zu Anfang die Möglichkeit gegeben, vom ATmega8 Meldungen am PC auszugeben, was natürlich auch die Fehlersuche erleichterte.

Bei der Procyon AVRlib ist die Lizenz zu beachten (in Kurzform: man muss Dritten auf Verlangen den gesamten Quelltext zur Verfügung stellen, falls Teile der Procyon Bibliothek genutzt werden). Die Procyon AVRlib ist unter dem Link:
http://hubbard.engr.scu.edu/embedded/avr/avrlib erhältlich.

5.4.3. Verwendete Dateien der Procyon AVRlib

Die Ausführbare Datei cmdlinetest.c verwendet folgende C-Dateien mit zugehörigen Header-Dateien aus der Procyon AVRlib:

a2d.c / a2d.h	Analog-Digital-Konverter Funktionen-Bibliothek
avrlibdefs.h	AVRlib globale Definitionen und Makros
avrlibtypes.h	AVRlib globale Typen und Typdefinitionen
buffer.c / buffer.h	Mehrzweck Buffer-Struktur und Funktionen
cmdline.c / cmdline.h	Kommandozeilen-Interface Bibliothek
cmdlineconf.h	Kommandozeilen-Interface Bibliothek-Konfiguration
pwmsw.c / pwmsw.h	Pulsweitenmodulation Funktionen-Bibliothek
rprintf.c / rprintf.h	Printf Ausgabe-Routinen

timer.c timer.h	Timer Funktionen-Bibliothek
uart.c / uart.h	UART Treiber
vt100.c / vt100.h	VT100 Terminal Funktionen-Bibliothek
global.h	Globale Header-Datei für AVRlib Projekte

Diese Dateien wurden für die Software dieser Arbeit als Grundlage verwendet. Weiter unten im Text werden sie einzeln genauer beschrieben.

5.5. Header-Datei global.h

Zu jedem Procyon Projekt gibt es eine global.h Header-Datei die sich im selben Verzeichnis wie die ausführbare Datei des Projektes befindet. In der Datei global.h, – und nur hier – wird die Taktgeschwindigkeit des Mikrocontrollers eingegeben.

5.6. Erzeugen von Maschinencode

Aus dem C-Quelltext erzeugt der AVR-GCC-Compiler Maschinencode für den Mikrocontroller. Dieser Code liegt dann im Intel Hex-Format vor ("Hex-Datei"). Die Programmiersoftware „AVRDude“ liest diese Datei und überträgt die enthaltene Information in den Speicher des Mikrocontrollers.

Um aus einem C-Code eine Hex-Datei zu erhalten, muss man den AVR-GCC-Compiler mit den richtigen Optionen aufrufen. Grundsätzlich stehen dazu zwei verschiedene Ansätze zur Verfügung:

Zum einen gäbe es hier die Verwendung einer Integrierten Entwicklungsumgebung. Das sind unter Umständen mehrere Programme, die gemeinsam die Hex-Datei erstellen. Da sich die verschiedenen Entwicklungsumgebungen in ihrer Bedienung stark unterscheiden und auch nicht für alle Plattformen zur Verfügung stehen, tendiere ich zur zweiten Möglichkeit eine Hex-Datei zu erstellen: Durch die Verwendung des Programms „Make“ mit passenden Make-Dateien (Makefile).

5.7. Makefile

Ein Makefile ist eine Textdatei, die festlegt, wie ein oder mehrere Programme kompiliert werden. Der Grundgedanke dabei ist, dass gewisse Dateien in Abhängigkeit zu anderen Dateien stehen. Wünschenswert ist daher eine Möglichkeit, die Abhängigkeiten zwischen den einzelnen Dateien (Header, Module, Bibliotheken, ausführbare Programme, . . .) einmal zu definieren, wobei dann bei einer Änderung einer Datei nur die Teile eines Projekts neu erstellt werden müssen, die laut Abhängigkeiten im Makefile auch von der Änderung betroffen sind.

Gibt man nun „make“ in die Bash bzw. Eingabeaufforderung ein, so wird das Programm make.exe, das beim Installieren von WinAVR in das Unterverzeichnis WinAVR\utils\bin kopiert wurde, aufgerufen. „Make“ sucht automatisch das Makefile im aktuellen Arbeitsverzeichnis, das dann die Ablaufsteuerung des Programms „Make“ darstellt.

Es werden viele verschiedenen Makefiles als Vorlage angeboten, die man für seine eigenen Bedürfnisse anpassen kann. Da Procyon auch eine solche Vorlage bietet, habe ich diese verwendet. Hier gibt es aber eine Besonderheit: Procyon verwendet zwei Makefiles, wobei das eine vom anderen eingebunden wird.
Das Makefile „avrproj_make“ ist hierbei ein allgemeines Makefile, das bei allen Procyon Projekten gleich ist und viele allgemeine Funktionen bietet. Das eigentliche Makefile „makefile“, enthält nun die speziellen Angaben zum Projekt, wie z. B. den verwendeten Mikrocontroller-Typ und die C-Dateien. Außerdem bindet dieses Makefile das allgemeine Makefiel „avrproj_make“ mit folgender Zeile ein:

```
include avrproj_make
```

makefile, Zeile 38

Beide Makefiles sind im gleichen Verzeichnis wie die C-Dateien des Projekts gespeichert, und haben keine Dateiendung.

5.8. Fuse-Bits programmieren

5.8.1. Externen Quarz verwenden

Auf Platine 1 wurde ein externer Quarz angebracht, der dem ATmega8 einen Takt von 8 MHz geben soll, jedoch wird er noch nicht genutzt. Noch immer arbeitet der ATmega8 mit seiner intern voreingestellten 1 MHz Taktfrequenz. Um dies umzustellen, muss man sogenannte Fues-Bits (Sicherungs-Bits) im ATmega8 programmieren. Diese Fuse-Bits beeinflussen das Verhalten des ATmega8 auf grundlegender Ebene. Mit ihnen kann man z. B. Ein-

stellungen für externe Taktgeber, Reset-Einstellungen, etc. beeinflussen. Deshalb ist beim Programmieren der Fuse-Bits besondere Vorsicht geboten, denn wenn hier falsche Einstellungen vorgenommen werden, kann es passieren, dass der ATmega8 von einer nichtvorhandenen Taktquelle ausgeht und keine Reaktion mehr zeigt. Der Mikrocontroller kann dann nicht mehr umprogrammiert werden, außer man stellt ihm die geforderte Taktquelle zur Verfügung.

Fuse-Bits können programmiert oder unprogrammiert sein. Eine 0 bedeutet hierbei „programmiert" und eine 1 „unprogrammiert". Dies kann verwirrend sein und leicht zu falschen Einstellungen führen, weshalb die Programmierung der Fuse-Bits besonderer Vorsicht bedarf.

An dieser Stelle ist der Fuse-Bits-Rechner von Mark Hämmerling, den er auf seiner Website anbietet, hilfreich. Er bietet die Möglichkeit, die zu programmierenden Bits auszuwählen, woraufhin dann der benötigte Hex-Wert berechnet wird.
Der Link zum „AVR Fuse Calculator" von Mark Hämmerling:
http://haemi.dyndns.org/local/fusecalc/calc.cgi?ID=atmega8

5.8.2. Fuse-Bits abfragen und programmieren

Um Fuse-Bits abzufragen und zu programmieren benutzt man die gleiche Software (AVRDude) und Hardware (ISP-Programmieradapter) wie zum Hochladen eines Programms in den Flash-Speicher des ATmega8.

Mit folgendem Befehl in der Bash bzw. Eingabeaufforderung kann man den hierzu benötigten Terminal-Modus von „AVRDude" starten.

```
avrdude –p m8 –c bsd –t
```

Ist der Teminal-Modus geöffnet, hat man unter Anderem Zugriff auf die drei Fuse-Bytes des ATmega8:

- Fuse Low Byte (lfuse)
- Fuse High Byte (hfuse)
- Extended Fuse Byte (efuse)

Für die Umstellung auf einen externen Quarz werden nur die beiden Fuse-Bytes hfuse und lfuse benötigt.

Bit-Nummer	lfuse	hfuse
7	BODLEVEL	RSTDISBL
6	BODEN	WDTON
5	SUT1	SPIEN
4	SUT0	CKOPT
3	CKSEL3	EESAVE
2	CKSEL2	BOOTSZ1
1	CKSEL1	BOOTSZ0
0	CKSEL0	BOOTRST

Tabelle 5.1: Die Bits von hfuse und lfuse

Gibt man nun den Befehl `dump lfuse` bzw. `dump hfuse` im Terminal-Modus ein, wird bei beiden Fuse-Bytes der aktuelle Wert der Fuse-Bytes ausgegeben. Hier zwei mal ff. Dies bedeutet, dass im Lieferzustand alle Fuse-Bits unprogrammiert sind.

```
dump lfuse
```

→ 0000 ff

```
dump hfuse
```

→ 0000 ff

Aus dem Datenblatt des ATmega8 ist zu entnehmen, dass bei Quarzen die maximale Frequenz 8 MHz sein darf, wenn CKOPT unprogrammiert (1) ist, und 16 MHz, wenn CKOPT programmiert (0) ist [ATmega8]. Bei einem 8 MHz Quarz sollte CKOPT somit unprogrammiert sein.

CKSEL0 sollte programmiert sein, und gibt die Art des Quarzes an.

CKSEL3, CKSEL2 und CKSEL1 sollten unprogrammiert sein und geben an, dass die Frequenz des Quarzes zwischen 3 und 8 MHz liegt.

SUT1 und SUT0 sollten unprogrammiert sein. Diese Fuse-Bits geben an, wie lange der Mikrocontroller beim Einschalten verzögert. Mit dieser Einstellung wurde die kürzeste Verzögerungszeit gewählt.

Um nun die Fuse-Bytes auf den externen Quarz einzustellen, programmiert man die Fuse-Bytes mit dem Befehl `write`:

```
write lfuse 0 0xfe
write hfuse 0 0xd9
```

5.9. Timer

Ein Timer ist ein Zeitgeber. Er kann dazu verwendet werden, Zeitspannen zu messen oder Zeitspannen bestimmter Länge zu erzeugen bzw. Spannungspulse bestimmter Längen zu erzeugen.

Der ATmega8 verfügt über drei Timer:
- Timer 0 (8-bit Timer)
- Timer 1 (16-bit Timer)
- Timer 2 (8-bit Timer)

Ein 8-bit Timer zählt aufwärts bis 255, um dann wieder bei 0 zu beginnen. Bei jedem Überlauf von 255 auf 0 wird ein Interrupt ausgelöst. Ist eine Interrupt-Routine gegeben, wird diese jedes Mal ausgeführt. Timer 1 und 2 sind besondere Timer, sie können z. B. für die *Pulsweitenmodulation* verwendet werden.

5.10. Vorteiler

Der Vorteiler dient dazu, den Takt des Mikrocontrollers, der bekanntlich vom externen Quarz kommt, um einen einstellbaren Faktor zu reduzieren. Die so geteilte Frequenz wird den Eingängen der Timer zugeführt. Bei Teilung 1 geht z. B. die volle Quarzfrequenz zum Timer. Bei Teilung 8 nur ein Achtel der Quarzfrequenz.

5.11. Interruptgesteuerter Programmablauf

Der Programmablauf der Software dieser Arbeit ist hauptsächlich interruptgesteuert. Das bedeutet, dass eine Endlos-Hauptschleife des Programms durch Interrupts unterbrochen wird, um Unterroutinen auszuführen. Eine Steuerung des Programmablaufs durch Interrupts ist effektiv und entlastet den Prozessor.

Jeder Unterroutine ist ein Interruptlevel zugeordnet, das die Priorität bestimmt. Eine Routine höherer Priorität kann eine Routine niedriger Priorität unterbrechen.

Interrupts können nicht nur durch einen Timerüberlauf ausgelöst werden, sondern auch durch besondere Ereignisse, wie z.B. einer Änderung an einem Pin oder einer abgeschlossenen Analog-Digital-Wandlung.

Funktionen zur Interrupt-Verarbeitung werden in den Include-Dateien interrupt.h derAVR-Libc zur Verfügung gestellt, die unter Anderem von der Datei timer.c eingebunden werden.

Interrupts können während des Programmablaufs mit Hilfe der Funktionen `sei()` und `cli()` ein- bzw. ausgeschaltet werden. Diese Funktionen haben Zugriff auf das „Global Interrupt Enable Bit“ im *Status Register (SREG)*.

5.11.1. In der Software verwendete Interrupts

5.11.1.1. Überlauf Timer 0

Der 8-bit Timer 0 wird beim Initialisieren in der Datei timer.c aktiviert und bekommt den Vorteiler 64 zugeteilt, der in der Header-Datei timer.h angegeben ist. Der 8-bit Timer 0 hat eine Auflösung von 256 und löst bei jedem Überlauf einen Interrupt aus.
In Anbetracht des Vorteilers wird somit alle 16384 (64 * 256) Quarztakte ein Timerüberlauf-Interrupt ausgelöst, der den normen Programmablauf unterbricht. Es wird dann die zugehörige Interrupt-Routine aufgerufen.

5.11.1.2. Externe Interrupts

Wie weiter oben bereits erklärt, hat das Wählsignal eine ganz bestimmte Form. In genauen Abständen sinkt das Signal von 5V auf 0V ab und zeigt somit die gewählte Ziffer an. Dieses Signal ist ideal für die Verwendung von externen Interrupts.

Das Signal wird hierbei an den Pin PD2 (INT0) des ATmega8 gelegt. Den ATmega8 kann man nun so einstellen, dass er bei jeder Signaländerung an diesem Pin einen Interrupt auslöst und auch hier eine passende Interrupt-Routine ausgeführt wird.

5.12. Pollingverfahren

Der Telefonhörerstatus wird per Pollingverfahren abgefragt. Das bedeutet, dass in festgelegten Zeitabständen der Telefonhörerstatus abgefragt und in einer Variable gespeichert wird. Damit diese Abfrage nicht zu häufig geschieht und den Programmablauf nicht allzu sehr stört, wird sie nicht in der Hauptschleife des Programms platziert, sondern in der Interrupt-Routine des Timers 0.

5.13. Watchdog-Timer

Falls sich das Programm einmal aufhängen sollte, bietet ein „Watchdog“ die Lösung. Der Watchdog enthält einen separaten Timer, der je nach eingestelltem Vorteiler von 0 hochzuzählen beginnt. Wenn nun eine vorher eingestellte Anzahl von Zyklen erreicht wurde, löst der Watchdog einen Reset des Programms aus.
Um im Normalbetrieb einen Reset zu verhindern, muss man den Watchdog-Timer regelmäßig wieder zurücksetzten. In der Datei cmdlinetest.c war noch kein Watchdog-Timer enthalten und wurde daher von mir eingefügt.

5.14. C-Dateien des Projekts

Prinzipiell werde ich nur die Quelltext-Bereiche in den C-Dateien von Procyon beschreiben, die entweder von mir geändert oder eingefügt wurden, oder für die Beschreibungen relevant sind. Der komplette Quelltext ist auf der beigefügten CD-ROM zu finden.
Die C-Datei teltools.c wurde extra für dieses Projekt erstellt. Sie enthält die wichtigsten Funktionen für den ATmega8, um mit dem Wählscheibentelefon kommunizieren zu können.

5.14.1. Datei cmdlinetest.c

Die Datei cmdlinetest.c ist die ausführbare Datei, in der sich die main-Funktion befindet.

5.14.1.1. Bibliotheken einbinden

In der Datei cmdlinetest.c und allen anderen C-Dateien werden zunächst einige Teile der Bibliotheken AVR-Libc und Procyon AVRlib eingebunden, damit vorgefertigte Funktionen wie z. B. die Timer, der Watchdog-Timer usw. im Programm genutzt werden können.

5.14.1.2. Interrupts deaktivieren

Die darauf folgende main-Funktion schaltet mit dem Funktionsaufruf

```
  cli();
```

cmdlinetest.c, Zeile 52

zunächst alle möglichen Interrupts ab. Diese Interrupts werden unter Anderem mit den nächsten Funktionsaufrufen initialisiert und sollen erst im späteren Verlauf der cmdlinetest.c-Datei mit dem Funktionsaufruf

```
  sei();
```

cmdlinetest.c, Zeile 138

aktiviert werden.

5.14.1.3. Initialisierung

Die folgenden Funktionsaufrufe in der Datei cmdlinetest.c initialisieren die wichtigsten Funktionen der Software (wie z. B. das UART, die Timer, den Analog-Digital-Wandler, etc.), indem sie die Initialisierungs-Funktionen in deren C-Dateien aufrufen.

Aber auch der Watchdog-Timer wird hier initialisiert. Der Funktionsaufruf `wdt_enable(WDTO_500MS)` bewirkt, dass ein Reset des Programms ausgeführt wird, wenn der Watchdog-Timer nicht innerhalb von 500 ms zurückgesetzt wird.

```
  // initialize the UART (serial port)
  uartInit();

  uartSetBaudRate(9600);

  // make all rprintf statements use uart for output
  rprintfInit(uartSendByte);

  // initialize pwm
  pwmInit();

  // initialize watchdog timer
  wdt_enable(WDTO_500MS);

  // initialize the timer system
  timerInit();
```

```
  // intitialize the analog-digital-converter
  adInit();

  // initialize vt100 terminal
  vt100Init();
```

cmdlinetest.c, Zeilen 57 - 78

5.14.1.4. Datenrichtung bestimmen

In der main-Funktion werden dann als nächstes die Ein- und Ausgangs-Ports des ATmega8 eingestellt. Jeder Port (B, C, D) wird über 3 Register gesteuert.

DDRx Datenrichtungsregister für Portx

x entspricht den Ports B, C, D des ATmega8. Mit diesem Register kann man den ganzen Port, oder einzelne Pins des ATmega8 als Eingang oder Ausgang definieren. Eine 0 im Register steht für Eingang, eine 1 für Ausgang.

PINx Eingangsadresse für Portx

Diese dient dazu, den Zustand des Port-Pins abzufragen. Die Bits in PINx entsprechen dem Zustand der Portpins. Wenn z. B. im Bit PINB0 eine 1 steht, dann ist der Pin HIGH, dort liegt sozusagen eine Spannung an, die entweder vom ATmega8 oder von außen kommt. Eine 0 steht hier für LOW.

PORTx Datenregister für Portx

Dieses Register wird verwendet, um die Ausgänge eines Ports anzusteuern. Bei den Pins können über PORTx die internen *Pull-Up-Widerstände* aktiviert oder deaktiviert werden (1 = aktiv). Somit kann der ATmega8 selber 5V an einem Pin anlegen.

Da der Pin 2 des Port B (PB2) der PWM-Ausgang OC1B ist, an dem der ATmega8 PWM-Signale ausgibt, wird dieser Pin hiermit

```
  DDRB = (1<<DDB2);
```

cmdlinetest.c, Zeile 84

als Ausgang definiert.

Pin 0 von Port B (PB0) wird hingegen als Eingang definiert, da an diesem Pin das Telefonhörerstatus-Signal von der Telefonhörerstatus-Schaltung angelegt wird, und der ATmega8 dieses Eingangs-Signal mit dem Pollingverfahren abfragen soll.

```
  DDRB &= ~(1<<DDB0);
```

cmdlinetest.c, Zeile 88

Der Gesamte Port C wird als Eingang definiert, da alle Ports für den Analog-Digital-Wandler als analoge Eingänge fungieren. Bisher wird allerdings nur der Pin PC0 (ADC0) als Eingang für ein analoges Signal verwendet.

```
  DDRC = 0x00;
```

cmdlinetest.c, Zeile 92

Der ATmega8 hat zwei Pins (PD2 und PD3), mit denen Interrupts ausgelöst werden können (in diesem Fall externe Interrupts genannt INT0 und INT1), wenn sich das Signal, das von außen kommt, an diesen Pins verändert.
Diese Eigenschaft ist z. B. für das Signal, das von der Wählimpuls-Schaltung kommt, hilfreich. Jedes Mal, wenn sich das Wählimpulssignal ändert, wird im ATmega8 ein Interrupt ausgelöst, womit dann die Wählimpulse gezählt werden können.
Das Wählimpulssignal wird an den Pin PD2 des Port D des Atmega8 gelegt. Hierfür muss der Pin PD2 allerdings als Ausgang definiert werden, damit Interrupts ausgelöst werden können. [ATmega8]

```
 DDRD |= (1<<PD2);
```

cmdlinetest.c, Zeile 96

Zuletzt wird noch der Pin 7 von Port D (PD7) als Ausgang definiert, weil an diesem Pin das Signal vom ATmega8 ausgesendet werden soll, das das Relais schaltet, wodurch das Telefon dann zu klingeln beginnt.

```
  DDRD = (1<<DDD7);
```

cmdlinetest.c, Zeile 100

5.14.1.5. Funktion goCmdline()

In der main-Funktion wird zuletzt die Funktion goCmdline(), die sich auch in cmdlinetest.c befindet, aufgerufen.

Beim Aufruf der Funktion goCmdline()wird zunächst alles, was im Terminal-Fenster am PC steht, gelöscht, und der Begrüßungstext „Welcome to VoIP FeTAp!“ ausgegeben. Daraufhin wird das Kommandozeilen-Interface initialisiert, indem die Funktion cmdlineInit(), die sich in der Datei cmdline.c befindet, aufgerufen wird.
Im weiteren Quelltext können Kommandos und deren zugehörigen Funktionen eingegeben werden. Wird nun ein Kommando in das Terminal-Fenster des PCs eingegeben, das hier bekannt ist, wird die zugehörige Funktion aufgerufen. Somit kann man über das Terminal-Fenster mit der Software auf dem ATmega8 kommunizieren.

An dieser Stelle wurden von mir die Kommandos „r“ und „p“ eingefügt. Das Kommando „r“ ruft eine Funktion ringBell() auf, die sich in der Datei teltools.c befindet, und das Telefon zum Klingeln bringt. Das Kommando „p“ gibt mit der zugehörigen Funktion print_receiver(), die sich auch in teltools.c befindet, den aktuellen Telefonhörerstatus aus.

Als nächstes werden die bereits initialisierten Interrupts mit dem Funktionsaufruf sei() aktiviert.

```
...
// print welcome message
vt100ClearScreen();
vt100SetCursorPos(1,0);
rprintfProgStrM("\r\nWelcome to VoIP FeTAp!\r\n");

// initialize cmdline system
cmdlineInit();

// direct cmdline output to uart (serial port)
cmdlineSetOutputFunc(uartSendByte);

// add commands to the command database
cmdlineAddCommand("exit",    exitFunction);
cmdlineAddCommand("r",       ringBell);
cmdlineAddCommand("p",       print receiver);

// send a CR to cmdline input to stimulate a prompt
cmdlineInputFunc('\r');
```

```
// set state to run
Run = TRUE;

// enable interrupts
sei();
...
```

cmdlinetest.c, Zeilen 115 - 138

Sehr wichtig ist die darauf folgende Endlosschleife in der Funktion `goCmdline()`, die nur noch durch Interrupts oder dem Kommando „exit“ im Termial-Fenster unterbrochen bzw. beendet werden kann. In dieser Endlosschleife wird ständig abgefragt, ob Daten vom PC an den ATmega8 geschickt worden sind, und ob diese ein vorher bekanntes Kommando darstellen. Zuletzt wird in der Endlosschleife der Watchdog-Timer mit Hilfe des Funktionsaufrufs `wdt_reset()` zurückgesetzt.

```
// main loop
while(Run)
{
   // pass characters received on the uart (serial port)
   // into the cmdline processor
   while(uartReceiveByte(&c))
   {
      cmdlineInputFunc(c);
   }

   // run the cmdline execution functions
   cmdlineMainLoop();

   // reset the watchdog
   wdt_reset();
}
```

cmdlinetest.c, Zeilen 140 - 155

5.14.2. Datei timer.c

In der Datei timer.c werden die Timer 0 und 1 sowie die Externen Interrupts verwaltet.

In der Funktion `timerInit()` werden zunächst drei verschiedene Funktionen aufgerufen, die die einzelnen Interruptquellen initialisieren.

```
void timerInit(void)
{
   // initialize extern interrupt INT0
   extInt();

   // initialize timer 0 and 1
   timer0Init();
   timer1Init();
   ...
}
```

timer.c, Zeilen 55 - 67

Zum einen ist das der externe Interrupt INT0. Wie weiter oben schon erklärt, soll durch ein externes Ereignis an Pin PD2 ein Interrupt ausgelöst werden. In der Funktion `extInt()`, die diesen Interrupt initialisiert, sind genauere Einstellungen enthalten. Mit dem Register MCUCR (MCU Control Register) kann man dem ATmega8 sagen, dass er nicht nur beim Ansteigen der Spannung an Pin PD2 von 0V auf 5V, sondern auch beim Absinken von 5V auf 0V einen Interrupt auslösen soll, was für das Zählen der Wählsignale hilfreich ist.
Setzt man das Bit INT0 im Register GICR (General Interrupt Control Register) auf 1, dann ist der externe Interrupt INT0 aktiviert. Ist aber das globale Interrupt Bit im *Status Register* (SREG) nicht gesetzt ,so ist dieser Interrupt trotzdem nicht aktiv.

```
void extInt(void)
{
   // trigger on any logical change on INT0
   MCUCR = 0;
   MCUCR |= (1<<ISC00);

   // enable INT0
   GICR |= (1<<INT0);
}
```

timer.c, Zeilen 70 - 78

Als nächstes werden die Timer 0 und 1 mit den Funktionen `timer0Init()` und `timer1Init()` initialisiert.

Dem Timer 0 wird in der Funktion `timer0Init()` der Vorteiler 64 zugeteilt, der in der Header-Datei timer.h eingetragen ist. Timer 0 zählt ein 8-Bit-Register (TCNT0) mit jedem Taktsignal um eins hoch. Dieses Register TCNT0 wird zu Anfang auf 0 zurückgesetzt.
Um nun Interrupts durch den Überlauf des Timer 0 bei 255 zu erlauben, muss das Bit TOIE0 im Register TIMSK gesetzt werden. Aber auch hier ist der Interrupt trotzdem nicht aktiviert, wenn das globale Interrupt-Bit im *Status Register* nicht gesetzt ist.

```
// 8-bit timter0 for telefone functions
void timer0Init(void)
{
   // set prescaler
   timer0SetPrescaler( TIMER0PRESCALE );

   // set counter1 to zero */
   TCNT0 = 0x00;

   // enable counter1 overflow interrupt*/
   TIMSK |= (1<<TOIE0);

   // initialize time registers
   timer0ClearOverflowCount();
}
```

timer.c, Zeilen 81 - 95

Timer 1 läuft im „Phase Correct, 8-bit PWM-Modus“, der durch das Setzten des Bits WGM10 im Register TCCR1A angegeben wird. Das gesetzte Bit COM1B1 im selben Register gibt an, dass das PWM-Signal am zweiten PWM-Ausgang OC1B (PB2) des ATmega8 ausgegeben wird. Außerdem wird hierdurch auch der „nicht invertierende PWM-Modus“ gewählt, der in der Erklärung der Datei pwmsw.c näher erläutert wird.

Durch das Setzen des Bits CS10 im Register TCCR1B erhält der Timer 1 den Vorteiler 1. Das bedeutet, dass die volle Quarzfrequenz von 8 MHz an Timer 1 weitergegeben wird.

Timer 1 wird in dieser Datei dann als nächstes auf 0 gesetzt. Und die Interrupts, die durch Timer 1 entstehen, werden zugelassen.

```
  // 16-bit timer1 for speech (adc, pwm)
  void timer1Init(void)
  {
    // PWM-Output-frequency =
    // (Quarz-Frequency/Timer Prescale) / (2^8-Bit-PWM *2)

    // PWM-Output-frequency = (8 000 000Hz/1) / ((2^8*2))= 15625 Hz

    // Timer 1, PWM Phase Correct, 8-bit
    TCCR1A |= (1<<WGM10) | (1<<COM1B1);

    /* tmr1 prescaler 1 */
    TCCR1B |= (1<<CS10);

    /* set counter1 to zero */
    TCNT1 = 0x00;

    /* clear counter1 overflow flag */
    TIFR |= (1<<TOV1);

    /* enable counter1 overflow interrupt*/
    TIMSK |= (1<<TOIE1);
}
```

timer.c, Zeilen 98 - 118

In der Funktion `timerInit()` werden nach der Initialisierung der Timer und Interrupts außerdem noch alle Interrupts mit ihren zugehörigen Funktionen mit Hilfe der Funktion `timerAttach()` verknüpft.

```
  ...
  timerAttach(EXTERN0INTERRUPT, extInt0);
  timerAttach(TIMER0OVERFLOW_INT, timerOverflow0);
  timerAttach(TIMER1OVERFLOW_INT, timerOverflow1);
  ...
```

timer.c, Zeilen 64 - 66

Wird nun z. B. von Timer 0 ein Interrupt ausgelöst, so wird zunächst die Interrupt-Routine TIMER_INTERRUPT_HANDLER(SIG_OVERFLOW0) in der Datei timer.c aufgerufen. In einer Interrupt-Routine darf auf Grund des zugehörigen Registers nur ein Befehl stehen. Mit diesem Befehl wird die vorher mit Timer 1 verknüpfte Funktion `timer0verflow0()` aufgerufen. Diese Funktion enthält nun alle wichtigen Aufgaben, die nach einem Interrupt des Timers 0 aufgeführt werden sollen.

Die Funktionen `timer0verflow0()` und `extInt0()` sind in der Datei teltools.c implementiert. Die Funktion `timer0verflow1()` für den Timer 1 befindet sich hingegen in der Datei pwmsw.c.

```
//! Interrupt handler for tcnt0 overflow interrupt
TIMER_INTERRUPT_HANDLER(SIG_OVERFLOW0)
{
  // if a user function is defined, execute it too
  if(TimerIntFunc[TIMER0OVERFLOW_INT])
    TimerIntFunc[TIMER0OVERFLOW INT]();
}
```

timer.c, Zeilen 179 - 185

Die Funktionen `disable_timer1()` und `enable_timer1()` sind dazu da, um den Timer 1 bei Bedarf ein- und auszuschalten. Der Timer 1 ist für den Analog-Digial-Wandler und für die Ausgabe des PWM-Signals zuständig. Wird mit dem Telefon kein Gespräch geführt, so werden diese Funktionen nicht benötigt. Um diese Funktionen bei Bedarf auszuschalten, wurden die Funktionen `disable_timer1()` und `enable_timer1()` eingefügt.

```
void disable timer1(void)
{
  /* disable counter1 overflow interrupt*/
  TIMSK &= ~(1<<TOIE1);
}

void enable_timer1(void)
{
  /* enable counter1 overflow interrupt*/
  TIMSK |= (1<<TOIE1);
}
```

timer.c, Zeilen 121 - 132

5.14.3. Analog-Digital-Wandler

Die analoge Sprache, die beim Hineinsprechen in den Hörer des Wählscheibentelefons entsteht, muss für die Weiterverarbeitung und das Versenden über das Internet digitalisiert werden. Der ATmega8 verfügt über einen Analog-Digital-Wandler (im Folgenden kurz ADC genannt) mit 6 Eingangskanälen (ADC0 bis ADC6). Hier werden analoge Signale in 10-bit digitale Werte umgewandelt, die vom Atmega8 interpretiert werden können.

Der ATmega8 hat beim Eingang des eigentlichen ADCs einen Analog-Multiplexer, der dazu benutzt wird, um zwischen den 6 analogen Eingängen zu wählen. Das bedeutet, dass man bis zu 6 analoge Signale in digitale umwandeln kann, allerdings nicht gleichzeitig, sondern zeitversetzt.

Das analoge Signal, das an einem ADC-Eingang anliegt, muss größer als 0V, und kleiner als die Referenzspannung VREF sein. Die Referenzspannung ist eine externe Spannung die am AREF-Pin des Mikrocontrollers zugeführt werden muss.

Das Bestimmen der digitalen Werte, die das analoge Signal repräsentieren, nennt man Quantisieren. Das analoge Signal wird dabei mit einer bestimmten Abtastfrequenz abgetastet. Beim Abtasten wird ein analoges Signal dem nächsten erlaubten digitalen Wert zugewiesen. Die Anzahl der digitalen Werte wird Auflösung genannt und ist immer limitiert, z. B. auf 1024 Werte bei einem 10-bit digitalen Signal. Deshalb gibt es bei der Quantisierung immer einen Verlust von Informationen.

5.14.4. Datei a2d.c

In der Datei a2d.c ist eigentlich nur die Funktion `adInit()` relevant, denn hier wird der ADC eingerichtet. Mit dem Register ADMUX (ADC Multiplexer Selection Register) kann man unter Anderem den gewünschten Eingangs-Pin des ATmega8 auswählen, an dem das analoge Sprachsignal, das digitalisiert werden soll, ankommt. In diesem Fall wurde der Eingang ADC0 gewählt.

Mit dem Setzen oder Löschen des Bits ADEN im Register ADCSRA (ADC Control and Status Register A) kann man den ADC entweder ein- oder ausschalten.

Im Register ADCSRA muss man außerdem noch andere Einstellungen für den ADC vornehmen. Wie z. B. die Betriebsart mit der der ADC laufen soll:

- Einfache Wandlung (Single Conversion)
  In dieser Betriebsart muss jede Wandlung neu gestartet werden.
- Frei laufend (Free Running)
  In dieser Betriebsart wird eine Wandlung nach der anderen durchgeführt.

Hier wurde der Modus der „einfachen Wandlung“ gewählt, eine neue Wandlung wird von der Software immer zu einem bestimmten Takt gestartet.

```
//! initialize a2d converter
void adInit(void)
{
  // ADC0 is analog input
  ADMUX = 0X00;

  // sampling frequency = 15625 Hz
  // x * 15625 Hz = 8 000 000 Hz    x = 512
  // a single conversion takes 14 ADC cycles
  // 13 * 32 = 416 --> one ad-conversion lasts 416 cycles

  ADCSRA |= (1<<ADPS0);     // prescaler: 32
  ADCSRA |= (1<<ADPS2);     // prescaler: 32

  ADCSRA |= (1<<ADEN);      // activate ADC

  //do a dummy readout first

  ADCSRA |= (1<<ADSC);      // do single conversion
  // wait for conversion done, ADIF flag active?
  while(!(ADCSRA & 0x10));
}
```

a2d.c, Zeilen 27 - 46

Mit den Bits ADPS0, ADPS1 und ADPS2 des selben Registers kann man den Teilungsfaktor zwischen der Quarzfrequenz und dem Eingangstakt des Analog-Digital-Wandlers einstellen.

Der in dieser Software benötigte Teilungsfaktor wird wie folgt berechnet:
Die Abtastfrequenz soll, genauso wie die PWM-Frequenz, die im nächsten Abschnitt erklärt wird, 15,625 kHz betragen.
Teilt man nun die Quarzfrequenz von 8 MHz durch die Abtastfrequenz 15,625 kHz, so ergibt sich die Zahl 512. Diese Zahl gibt an, dass bei der gewünschten Abtastfrequenz von 15,625 kHz alle 512 Quarztakte eine Analog-Digital-Wandlung gestartet werden muss.
Aus dem Datenblatt des Atmeag8 [ATmega8] ist ersichtlich, dass alle Analog-Digital-Wandlungen (bis auf die aller erste, die dauert 25 Takte) 13 Takte dauern. Für die Berechnung des Teilungsfaktors bedeutet dies, dass man die 512 Quarztakte (die für eine Analog-Digital-Wandlung zur Verfügung stehen) durch die 13 Takte (die für eine Wandlung benötigt werden) teilt. Daraus ergibt sich die Zahl 39. Als ADC-Teilungsfaktor wird somit der nächstgelegene niedrigere mögliche Teilungsfaktor von 32 gewählt.

Multipliziert man nun die Takte, die für eine Wandlung nötig sind (13) mit dem ADC-Teilungsfaktor (32), dann ergibt sich die Anzahl der Quarztakte, die für eine Analog-Digital-Wandlung tatsächlich gebraucht werden (416). Die Berechnung des Teilungsfaktors ist korrekt, da der Wert 416 unterhalb der für eine Wandlung zur Verfügung stehenden 512 Quarztakte liegt.

In der Funktion `adInit()` wird zum Schluss lediglich die aller erste Analog-Digital-Wandlung angestoßen, da diese einige ADC-Takte länger dauert als die folgenden.

Die nächsten Analog-Digital-Wandlungen werden dann in der Datei pwmsw.c angestoßen. Auch deren Ergebnisse werden dort ausgelesen.

5.14.5. *Pulsweitenmodulation*

Das Sprachsignal, das vom Gesprächspartner „aus dem Internet“ kommt, soll über den ATmega8 an das Wählscheibentelefon geschickt werden. Somit muss der ATmega8 die digitale Sprache für das Wählscheibentelefon wieder analogisieren. Da für den ATmega8 kein eingebauter Digital-Analog-Wandler existiert, muss diese Funktion durch eine Hardware- oder Software-Lösung realisiert werden.

Die *Pulsweitenmodulation* (PWM) hat sich hierfür etabliert und bietet auch in diesem Fall die Lösung. Das digitale Signal wird analogisiert, indem an einem Ausgabe-Pin des ATmega8 Impulse mit voller Spannung und variabler Breite gesendet werden, die dann mit der Lautsprecher-Filter-Schaltung geglättet werden, woraus sich dann ein analoges Signal ergibt. [Mikro]

Grundsätzlich gibt es zwei Arten von PWM [Robo]:

- Software PWM
  Vorteil: Das PWM kann man auf jedem Ausgabe-Pin des Mikrocontrollers anwenden.
  Nachteil: Das PWM wird per Software generiert und ist daher eher langsam.

- Hardware PWM
  Vorteil: Dieses PWM ist sehr schnell, da die drei möglichen Hardware-PWM-Ausgänge nach Definition der Register/Werte selbständig ablaufen, und somit belastet das Hardware-PWM die Abarbeitung des Programms nicht.
  Nachteil: Die drei Hardware-PWM-Ausgänge sind bestimmten ATmega8 Ausgangs-Pins zugeordnet und können somit nicht auf beliebige Pins gelegt werden.

An dieser Stelle habe ich mich für das schnelle Hardware-PWM entschieden, da es die Abarbeitung des Programmablaufs nicht belastet und die Hardware PWM-Pins auch nicht für einen anderen Zweck benötigt werden.

5.14.5.1.Genaue Erklärung der Hardware-*Pulsweitenmodulation*

Da der ATmega8 ein rein digitales Bauteil ist, kann man ein Ausgangs-Pin entweder auf HIGH setzen, worauf am Ausgang die Versorgungsspannung 5V liegt, oder aber den Ausgang auf LOW setzen, wonach dann 0V anliegen. Bei der *Pulsweitenmodulation* wird nun periodisch mit einer festen Frequenz zwischen 5V und 0V umgeschalten. Dabei entsteht ein Rechtecksignal.

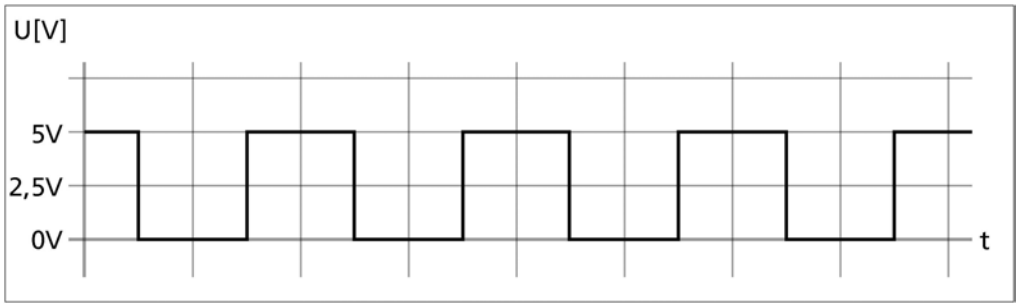


Abbildung 5.1.: PWM-Rechtecksignal

Für das Erzeugen einer periodischen Änderungen eines Signals können für die drei PWM-Ausgänge (PB1 (OC1A), PB2 (OC1B), PB3 (OC2)) des ATmega8 die Timer 1 und 2 verwendet werden.
Timer 1 ist ein 16-bit Zähler und kann daher zwei PWM-Ausgänge OC1A und OC1B verwalten, die allerdings gleiche Timer-Einstellungen haben und somit auch den gleichen Vorteiler. Timer 2 ist ein 8-bit Zähler und verwaltet den dritten PWM-Ausgang OC2.

In der Datei timer.c wurde der Timer 1 bereits für den OC1B-PWM-Ausgang initialisiert. Er wurde so eingestellt, dass er im 8-bit „nicht invertierenden PWM-Modus“ betrieben wird. Er zählt von 0 aufwärts bis zu Obergrenze (bei 8-bit 255) und danach wieder zurück auf 0, und wird als sogenannter Auf- und Ab-Zähler betrieben.

Der „nicht invertierende PWM-Modus“ bewirkt nun, dass der Ausgangspin OC1B beim Hochzählen des Zählers gelöscht und beim Herunterzählen gesetzt wird. In der Abbildung 5.2. ist das sehr gut erkennbar. Das Verhältnis zu Setzen und Löschen des Ausgangspins nennt man Tastverhältnis. Einmal Hoch- und Runterzählen ergibt dabei die Periode. Die Periode ist gleich der PWM-Ausgangsfrequenz vom PWM-Signal.

Die PWM-Ausgangsfrequenz berechnet sich wie Folgt:
PWM-Ausgangsfrequenz = (Quarzfrequenz / Vorteiler) / (Timerauflösung * 2)

Quarzfrequenz = 8 MHz
Vorteiler = 1
8-bit Timer 1 Auflösung = 256

PWM-Ausgangsfrequenz = (8 000 000 Hz / 1) / (256 * 2) = 15,625 kHz

Diese 15,625 kHz werden benötigt, denn die PWM-Ausgangsfrequenz muss doppelt – am besten vier mal – so groß sein, wie die Signalfrequenz, die nach dem Lautsprecher-Filter übrig ist. Nach der Lautsprecher-Schaltung hat das Signal 4 kHz.

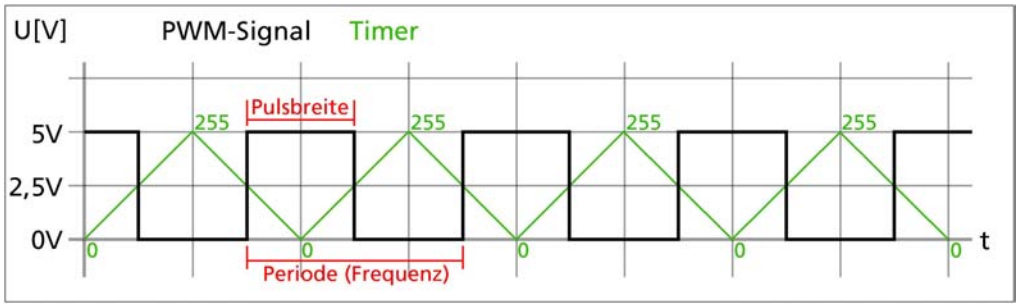


Abbildung 5.2.: Nicht invertierende PWM mit Tastverhältnis 50%

Um genau festzulegen, wann der Pin beim Hochzählen gelöscht und beim Herunterzählen gesetzt wird, benötigt man noch den Vergleichswert, der in das 16-Bit OCR1B (Timer Output Compare Register) Register geschrieben wird.

Überall dort, wo der Timer 1 diese Vergleichs-Linie schneidet, schaltet der Ausgang beim Hochzählen des Zählers auf LOW und beim Herunterzählen auf HIGH.
Folgende Abbildung soll den Zusammenhang zwischen dem Vergleichswert und dem generierten PWM-Signal aufzeigen.

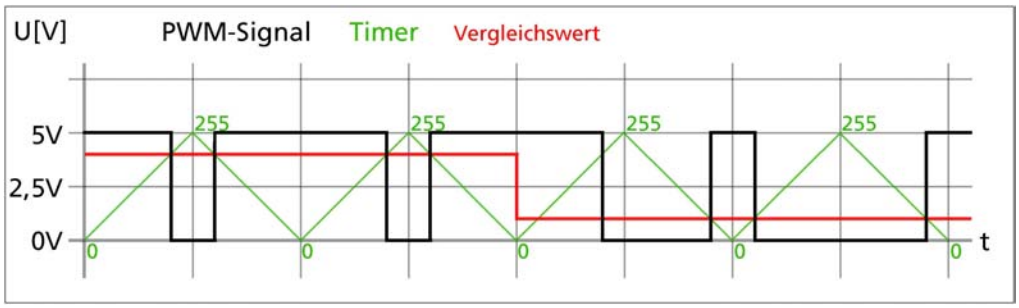


Abbildung 5.3.: Nicht invertierende PWM mit Vergleichswert

In Abbildung 5.3. ist zunächst ein Tastverhältnis von 80% und dann ein Tastverhältnis von 20% gezeigt. Ein Tastverhältnis von 80% erhält man, wenn der Vergleichswert im Vergleichsregister OCR1B den Wert 205 hat. Wenn der Timer z. B. beim Hochzählen den Wert 205 erreicht, so schaltet der PWM-Ausgang auf LOW. Ein Tastverhältnis von 20% erhält man hingegen, wenn der Vergleichswert den Wert 51 hat.

Wird am Telefonhörer nicht gesprochen, so hat das Rechtecksignal ein Tastverhältnis von 50%, d. h. dass der PWM-Ausgangs-Pin periodisch gleich lang auf HIGH und auf LOW gesetzt wird (siehe Abbildung 5.2.).
Dieses Rechecksignal wird an die Lautsprecher-Filter-Schaltung weitergeleitet. Der Filter glättet das Signal insofern, dass er die hohe PWM-Frequenz herausfiltert und in diesem Fall dann die halbe Betriebsspannung von 2,5V Gleichspannung übrig bleibt. Das Sprechen in den Telefonhörer bewirkt eine Änderung des Tastverhältnisses. Diese Änderung wird vom Filter nicht herausgenommen und ergibt dann das erwünschte analoge Sprachsignal.

Ein Sprachsignal hat seine Hauptinformationen unter 3 kHz. Somit ist die Grenzfrequenz fg des Lautsprecher-Filters auf 4 kHz gesetzt. Hierbei werden alle Frequenzen, die niedriger als die Grenzfrequenz fg sind durchgelassen. Alle Frequenzen, die höher als die Grenzfrequenz fg sind (wie die PWM-Frequenz), werden vom Lautsprecher-Filter herausgefiltert.

5.14.6. Datei pwmsw.c

Der Timer 1, der immer von 0 auf 255 zählt und dann wieder zurück auf 0, ist für das Erzeugen vom PWM-Signal notwendig. Timer 1 wurde bereits in der Datei timer.c initialisiert.

In der Funktion `timer0verflow1()`, die Timer 1 alle 512 Takte aufruft, wird zunächst der digitale Wert der letzten Analog-Digital-Wandlung direkt in das Vergleichsregister OCR1B geschrieben. Daraufhin wird eine neue Analog-Digital-Wandlung angestoßen, die bis zum nächsten Aufruf der Funktion `timer0verflow1()` abgeschlossen ist.

Das 10-bit Ergebnis der Analog-Digital-Wandlung berechnet man, mit der Formel:
ADC - ((VREF * ADC) / 2^10)

Der daraus entstandene Wert hängt somit von der Referenzspannung VREF, die am Pin AREF anliegt ab. Außerdem wird in dieser Formel berücksichtigt, ob es ein 10-bit oder 8-bit ADC-Ergebnis ist.

```
// ADC/PWM, every 512 cycles
void timerOverflow1(void)
{
   // 10-bit digital speech-signal
   // U = (VREF * ADC) / 1024
   OCR1B = ADC -((2,45 * ADC) / 1024);

   // conversion should be done, but
   // wait for conversion done, ADIF flag active?
   while(!(ADCSRA & 0x10));

   // start new single conversion
   ADCSRA |= (1<<ADSC);
}
```

pwmsw.c, Zeilen 45 - 58

Die Funktion `pwmInit()` in der Datei pwmsw.c setzt beim Initialisieren des PWMs lediglich das Vergleichsregister OCR1B auf 0.

```
void pwmInit(void)
{
    /* set PWM value to 0 */
    OCR1B = 0;
}
```

pwmsw.c, Zeilen 38 - 42

5.14.7. Datei teltools.c

Wird im Terminal-Fenster der Buchstabe „r“ eingegeben, so wird die zu diesem Kommando zugehörige Funktion `ringBell()` aufgerufen, die sich in der Datei teltools.c befindet. In dieser Funktion wird der interne *Pull-Up Widerstand* des Pins PD7 über das Register PORTD eingeschaltet und der Pin somit auf VCC gezogen. Der Pin PD7 schickt dann 5V an das Relais, dass mit dem Pin PD7 verbunden ist. Das Relais lässt die Telefonklingel ertönen. Dass das Telefon gerade klingelt, wird in der Variablen `ring` festgehalten.
Ist der Telefonhörer beim Aufrufen der Funktion `ringBell()` abgehoben, so wird die Telefonklingel nicht eingeschaltet, sondern im Terminal-Fenster die Meldung ausgegeben, dass das Telefon gerade belegt ist.

```
void ringBell(void)
{
  if(telreceiver == 0)  // if tel receiver down
  {
    countd = 0;         // correct start of bell-ring
    PORTD |= (1<<PD7); // set Pin PD7 high (ring the telephone bell)
    ring = 1;           // status: bell rings
  }
  else                  // if tel receiver up
  {
    rprintf("telephone busy!\r\n");
  }
}
```

teltools.c, Zeilen 45 - 57

Als nächstes folgt die Funktion `timerOverflow0()`. Sie wird jedes Mal aufgerufen, wenn Timer 0 einen Überlauf hat. In dieser Funktion werden bei jedem Aufruf einige Zähler (`counta` bis `countd`) um eins hochgezählt. Diese Zähler werden für die verschiedensten Aufgaben benötig.

```
  // Timer 0
  // function 488 times called in a second
  void timerOverflow0(void)
  {
    cli();                 //disable interrupts

    counta++;
    countb++;
    countc++;
    countd++;

    if (countc == 100)     // polling
    {
      tel_receiver();      // check tel receiver status
      countc = 0;
    }
      ...
}
```

teltools.c, Zeilen 61 - 77

`countc` wird z. B. verwendet, um regelmäßig den Telefonhörerstatus abzufragen, und ihn in der Variablen `telreceiver` zu speichern. Dabei wird, wenn `countc` bis 100 gezählt hat, die Funktion `tel_receiver()` aufgerufen, die sich ebenfalls in teltools.c befindet. Mit dieser Funktion wird abgefragt, ob an dem Pin PB0 5V anliegen oder nicht.

Liegen 5V an, ist der Telefonhörer abgehoben. Zeigt die Variable `telreceiver` dabei noch an, dass der Telefonhörer aufliegt, dann ist der Telfonhörer gerade eben abgehoben worden und es ist somit sinnvoll zu überprüfen, ob die Telefonklingel läutet. Gibt die Variable `ring` nun an, dass das Telefon klingelt, wird das Klingeln abgeschaltet und die Sprachübertragung wird eingeschaltet. Hierbei wird der Timer 1 mit dem Funktionsaufruf `enable_timer1()` eingeschaltet. Der Analog-Digital-Wandler und die *Pulsweitenmodulation* werden so ermöglicht.

Ist der Telefonhörer aufgelegt, wird der Timer 1 deaktiviert. Außerdem wird überprüft, ob kurz vorher Ziffern am Wählscheibentelefon gewählt wurden, die verworfen werden müssen.

Der durch die Funktion `tel_receiver()` erhaltene Telefonhörerstatus wird immer in der Variablen `telreceiver` festgehalten.

```
void tel_receiver(void)
{
  cli();                    // disable interrupts

  if((PINB & (1<<PINB0)))   // if tel receiver up
  {
    if(telreceiver == 0)    // if tel receiver was recently down
    {
      if(ring == 1)         // if tel bel rings
      {
        enable_timer1();    // start conversation
        PORTD &= ~((1<<PD7)); // stop tel bell
        ring = 0;
      }
    }
    telreceiver = 1;        // status: tel receiver up
  }
  else                      // if tel receiver down
  {
    disable_timer1();       // stop conversation

    // if dial-action active
    if ((y != 0) || (number_complete == 1))
    {
      y = 0;                // exit number-dialing
      number_complete = 0;
      cmdlinePrintPrompt();
    }
    telreceiver = 0;        // status: tel receiver down
  }

  sei();                    // enable interrupts
}
```

teltools.c, Zeilen 134 - 166

Der Zähler `countd` in der Funktion `timerOverflow0()` ist dafür zuständig, dass das Telefon in einem bestimmten Rhythmus klingelt. Ist nun das Telefonklingeln eingeschaltet, soll dass Telefon genau 2 Sekunden klingeln und danach 1 Sekunde still sein um wieder 2 Sekunden zu klingeln.

```
  // Timer 0
  // function 488 times called in a second
  void timerOverflow0(void)
  {
     ...

     if(ring == 1)
     {
       if((PIND & (1<<PIND7)))       // if tel bell rings
       {
         if(countd > 976)            // 2 seconds
         {
           PORTD &= ~((1<<PD7));     // stop tel bell
           countd = 0;
         }
       }
       if(!(PIND & (1<<PIND7)))      // if tel bell not rings
       {
         if(countd > 488)            // 1 second
         {
           PORTD |= (1<<PD7);        // ring tel bell
           countd = 0;
         }
       }
     }
        ...
}
```

teltools.c, Zeilen 80 - 98

Das letzte Stück Quelltext in der Funktion `timerOverflow0()` wird später in einem anderen Zusammenhang noch erläutert.

Die Funktion `extInt0()` wird bei einem externen Interrupt aufgerufen. Dafür muss sich das Signal am Pin PD2 des ATmega8 verändern. An diesem Pin ist das Wählsignal angeschlossen. Möchte man eine Nummer wählen, hebt man den Telefonhörer ab. Dabei ändert sich das Wählsignal das erste Mal und löst einen Interrupt aus. In der Funktion wird dann zunächst mit Hilfe von `counta`, `count1` und `count2` berechnet, wie lange der letzte externe Interrupt her ist. Liegt er länger als 50 Zähler-Einheiten zurück, kann es sein, dass eine neue Ziffer gewählt wird. Diesen Zustand hält man mit den Variablen `dial1` und `dial2` fest. Tritt der nächste Interrupt innerhalb von 15 bis 35 Zähler-Einheiten ein, so wird der erste Wählimpuls mit der Variablen `dialcount` gezählt. So lange der Telefonhörer abgehoben ist, und die Ziffer, die durch eine bestimmte Anzahl von Wählimpulsen dargestellt wird, noch nicht komplett ist, wird auch bei allen weiteren Interrupts überprüft, ob die Zeit seit dem letzten Interrupt zwischen 15 und 35 Zähler-Einheiten liegt. Wenn ja, wird `dialcount` immer um eins hochgezählt. Ist die erste Ziffer am Telefon fertig gewählt, so kann der Atmega8 anhand von `dialcount` diese Ziffer rekonstruieren.

```
// dialing signals
void extInt0(void)
{
  cli();                    //disable interrupts

  count1 = counta;          // x = time between the last interrupt
  x = count1 - count2;

  // if tel receiver up and dialing number not complete
  if ((telreceiver == 1) && (number_complete == 0))
  {
    if ((x == 0) | (x == 1))       // no dial-action
    {}

    if (dial1 == 1)         // possibly a dial action
    {
      if (dial3 == 1)       // dial-action active
      {
        if ((x > 15) & (x < 35))  // the next dial-impulse
        dialcount++;
        countb = 0;
      }
      if (dial2 == 1)       // is it the first dial-impuls?
      {
        if ((x > 15) & (x < 35))
        {
          dialcount++;
          dial2 = 0;        // status now: first dial-impulse exists
```

```
            dial3 = 1;        // status now: dial-action active
            countb = 0;
          }
        }
      }

      if (x > 50)             // possibly a dial-action starts
      {
        dial1 = 1;            // possibly a dial-action starts
        dial2 = 1;            // possibly the first dial-impuls
      }
    }

    count2 = count1;          // store counter to claculate the next

    sei();
    }
```

teltools.c, Zeilen 170 - 214

Die externen Interrutps werden wie in folgender Abbildung gezählt:

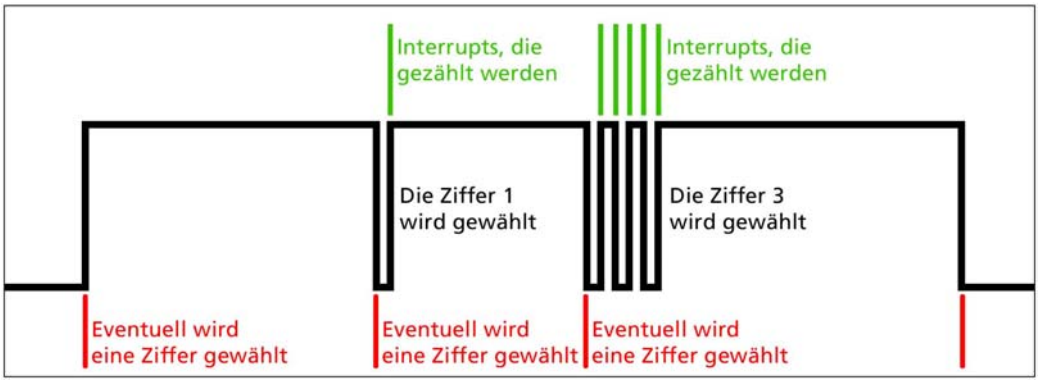


Abbildung 5.4.: Zählen der externen Interrupts

Die Variablen `dial1`, `dial2` und `dial3` dienen beim Zählen der Wählimpulse der Zustandbeschreibung:
Wenn die Möglichkeit besteht, dass aktuell eine Ziffer am Telefon gewählt wird, dann werden `dial1` und `dial2` auf 1 gesetzt. Wenn sich die Vermutung bestätigt, wird `dial2` wieder auf 0 gesetzt. `dial2` steht für den allerersten Wählimpuls. `dial3` wird gleichzeitig auf 1 gesetzt, was anzeigt, dass definitiv eine Ziffer gewählt wird. Dieses Verfahren dient

dazu, dass in der Funktion `extInt0()` immer in die Richtige if-Bedingung gesprungen wird.

Jedes Mal wenn `dialcount` um eins hochgezählt wird, wird auch der Zähler `countb` auf null gesetzt. Ist das Wählen einer Ziffer abgeschlossen, erreicht der Zähler `countb` bald den Wert 50. Ob der Zähler `countb` den Wert 50 erreicht hat, wird in der Funktion `timerOverflow0()` abgefragt.

Hat nun `countb` den Wert 50 erreicht, kann das bedeuten, dass das Wählen der ersten Ziffer abgeschlossen ist, und diese abgelesen werden kann. Fragt man zusätzlich die Variable `dial3` ab, und ist diese auf 1, so zeigt dies eindeutig an, dass gerade eine Ziffer gewählt wurde. Die gewählte Ziffer kann dann mit der Funktion `print_number()` am Terminal-Fenster ausgegeben werden. Alle Variablen, die für das Wählen zuständig sind, werden an dieser Stelle zurückgesetzt. Hier wird dann die Variable `y` aktiv, sie zählt die gewählten Ziffern und lässt nur die 12 Ziffern, die für eine IP-Adresse benötigt werden, zu. Sind dann alle Ziffern für die IP-Adresse gewählt, so gibt das Terminal-Fenster aus, dass die Nummer komplett ist. Es ist dann auch nicht mehr möglich weitere Ziffern hinzuzufügen. Sobald der Telefonhörer aufgelegt wird, wird die Variable `y` wieder auf 0 gesetzt. Hebt man dann den Telefonhörer wieder ab, kann von Neuem ein Wählvorgang beginnen.

```
// Timer 0
 // function 488 times called in a second
 void timerOverflow0(void)
 {
  …

   // possibly the end of dialing a number
 if (countb == 50)
 {
   if (dial3 == 1)         // end of dialed number
   {
     print_number();       // print the dialed number
     dial1 = 0;            // reset
     dial3 = 0;            // reset
     dialcount = 0;        // reset
     y++;

     if ((y == 3) || (y == 6) || (y == 9))
     {
       rprintf(" ");
     }

     if (y == 12)          // dialed number complete
```

```
      {
        rprintf("\r\nnumber complete!\r\n");
        cmdlinePrintPrompt();
        number_complete = 1;
        y = 0;

        // send dialed number to pc ...connecting...
      }
    }
  }

  sei();                    // enable interrupts
}
```

teltools.c, Zeilen 101 - 130

```
  // print dialed number
  void print number(void)
  {
    cli();                //disable interrupts

    if (dialcount == 1)
    {
      rprintf("1");       // the dialed number is 1
    }
    if (dialcount == 3)
    {
      rprintf("2");       // the dialed number is 2
    }
       ...

    sei();                // enable interrupts
  }
```

teltools.c, Zeilen 217- 264

Die letzte Funktion in der Datei teltools.c `print_receiver()` gibt immer den aktuellen Telefonhörerstatus im Terminalfester aus, wenn man das Kommando „p“ eingibt.

```
// print tel receiver status, if you write an p at comandline
void print_receiver(void)
{
  if(telreceiver == 0)    // tel receiver down
  {
    y = 0;                // new dialing-action
    rprintf("down\r\n");
  }
  else                    // tel receiver up
  {
    y = 0;                // new dialing-action
    rprintf("up\r\n");
  }
}
```

teltools.c, Zeilen 267 - 280

5.15. Batch-Datei

Nachdem nun alle relevanten Teile der Software erstellt wurden, geht es an das Übersetzen dieser Software in Maschinencode, damit sie auf den ATmega8 geladen werden kann.

Um das Kompilieren der C-Dateien und das Übertragen der Hex-Datei zum ATmega8 zu automatisieren, habe ich eine Batch-Datei „voip.bat“ erstellt. Beim Ausführen dieser Datei werden die Anweisungen darin nacheinander (Stapelverarbeitung) vom Betriebssystem ausgeführt.

Die auszuführenden Befehle werden beim Erzeugen einer Batch-Datei in eine einfache Textdatei geschrieben und mit der Dateiendung .bat abgespeichert.

Die Batch-Datei für diese Arbeit enthält folgendes:

```
make clean
make
avrdude -p m8 -c bsd -e -U flash:w:cmdlinetest.hex
```

voip.bat

Was mit dem Befehl `make clean` passiert, ist im Makefile definiert und bewirkt, dass alle temporären Dateien, die beim letzen Kompilieren entstanden sind, sicherheitshalber gelöscht werden. Durch den Befehl `make` wird das Makefile aufgerufen, und das Programm nach den darin enthaltenen Kompilier-Anweisungen kompiliert. Die dritte Zeile überträgt die gerade entstandene Hex-Datei mit dem Programm „AVRDude“ auf den ATmega8.

6. VoIP FeTAp in der Praxis

Wie das VoIP FeTAp in der Praxis funktioniert, soll mit den Folgenden Ablaufprotokollen aufgezeigt werden.

6.1. Ablaufprotokoll beim Anrufen

Beim Wunsch, einen Gesprächspartner anzurufen, hebt der Benutzer den Telefonhörer ab. Dadurch ändert sich das Signal, das von der Telefonhörerstatus-Schaltung an den ATmega8 geleitet wird. Der ATmega8 „weiß" nun, dass der Telefonhörer abgehoben ist. Das Kommando „p" im Terminal-Fenster gibt immer den aktuellen Telefonhörerstatus aus.

Wird in diesem Moment vom PC die Meldung an den ATmega8 geschickt, dass jemand dieses Telefon anrufen möchte (dieser Fall wird mit dem Kommando „r" dargestellt, dieses Kommando soll das Telefon klingeln lassen), so wird im Terminal-Fenster ausgegeben, dass ein Anruf zur Zeit nicht möglich und das Telefon belegt ist.

Der Benutzer wählt nun die IP-Adresse des gewünschten Gesprächspartners, dabei ändert sich das Signal, das von der Wählsignal-Schaltung zum Atmega8 gleitet wird. Der ATmega8 kann aus diesem Signal erkennen, welche Ziffern der Benutzer wählt, und gibt diese an den PC weiter, der sie im Terminal-Fenster ausgibt. Bei der IP-Adresse sind 4 mal 3 Ziffern möglich und werden auch so in Gruppen im Terminal-Fenster ausgegeben. Sind alle Ziffern gewählt, gibt der Mikrokontroller eine Meldung aus.
Hat sich der Benutzer verwählt, so legt er den Telefonhörer auf. Dabei werden die bisher gewählten Zahlen verworfen und er kann von neuem mit dem Wählen beginnen.

Abbildung 6.1.: Terminal-Fenster beim Anrufen

6.2. Ablaufprotokoll beim Angerufenwerden

Beim Angerufenwerden wird vom PC ein Signal an den Atmeag8 verschickt. In dieser Arbeit wird dieses Signal mit dem Kommando „r“ im Terminal-Fenster dargestellt. Der ATmega8 erkennt dieses Kommando und sendet daraufhin ein 5V Signal an die Telefon-Schaltung. Diese 5V bewirken, dass einigen Teilbereichen der Telefon-Schaltung eine Wechselspannung von 40V zugeführt wird, und die Telefonklingel zu läuten beginnt.

Hebt der Angerufene den Telefonhörer ab, ändert sich das Signal, das von der Telefonhörerstatus-Schaltung an den ATmega8 geleitet wird. Der ATmega8 erkennt dieses Signal und ermöglicht die Sprachübertragung. In dieser Arbeit wird die analoge Sprache vom Atmega8 digitalisiert und gleich darauf wieder analogisiert und an einem Lautsprecher, der an die Schaltung angeschlossen werden kann, ausgegeben.

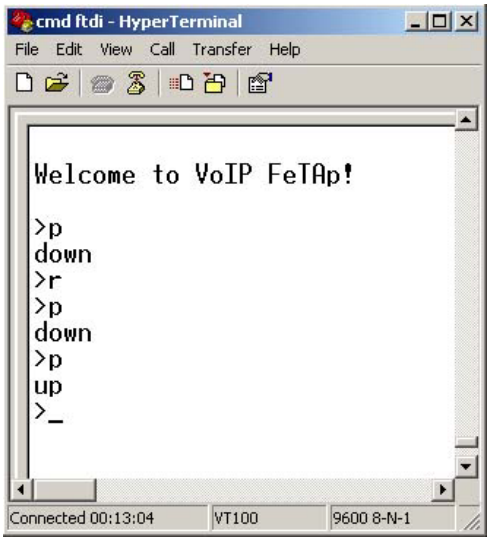


Abbildung 6.2.: Terminal-Fenster beim Angerufenwerden

7. Überlegungen zur Weiterentwicklung

In diesem Kapitel werden Überlegungen angestellt, wie dieses Projekt weitergeführt werden könnte.

7.1. Treiber für PC

Der PC sollte einen eigenen Treiber erhalten, damit er die Signale des ATmega8 nicht nur über das Terminal-Programm versteht. Bei der Übertragung der Sprache im Speziellen muss die Treibersoftware am PC per Pollingverfahren abfragen, ob der Puffer im Speicher des ATmega8 voll ist. Ist er voll, wird der Inhalt des Puffers per USB an den PC übertragen.
So erhält der PC 10-bit Rohdaten.

7.2. Soundbibliothek

Um die Soundrohdaten für die VoIP-Übertragung aufzubereiten, kann die Soundbibliothek „Open Source Audio Library Project“ zu finden unter folgendem Link hilfreich sein:
http://osalp.sourceforge.net/
.

7.3. VoIP

Der Begriff „Voice over IP“ (VoIP) bezeichnet die Technik, mit der Sprache über IP-Netzwerke übertragen werden kann. Um eine IP-Sprachübertragung zu ermöglichen, sollte überprüft werden, ob sich dafür eine bereits bestehende VoIP-Software eignet.

Die zwei bekanntesten Protokoll-Standards bei der IP-Sprachübertragung sind:
- H.323
- SIP

Diese Protokolle sind dazu da, um zunächst mit Hilfe der Adresse des Gesprächspartners (ähnlich Email-Adresse) die aktuelle IP-Adresse herauszufinden und daraufhin eine Verbindung aufzubauen. Im Folgenden werden diese beiden Protokolle, bezüglich der für diese Arbeit relevanten Unterschiede, beschrieben.

7.3.1. Protokoll H.323

Beim H.323-Protokoll wird für den Verbindungsausbau ein Gatekeeper verwendet. Zu seinen Aufgaben gehören Dienste im Bereich der Anrufsignalisierung und -steuerung, Umsetzung von Adressen, Zugriffs- und Bandbreiten-Kontrolle. Die Kommunikation wird bei diesem Protokoll somit vom Gatekeeper gesteuert. [ELKO]

VoIP-Software, die den H.323-Standard verwenden:
- NetMeeting
- GnomeMeeting
- OhPhone

7.3.2. Protokoll SIP

Der große Vorteil von SIP (Session Initiation Protokoll) ist, das dieses Protokoll alle Dienste wie HTTP, SMTP, MIME, URL und DNS mit nutzt. Statt einem Gatekeeper wird bei diesem Protokoll ein Proxy-Server verwendet. Bei diesem Protokoll steuert der Client die Kommunikation [ELKO]. Für das eigentliche Telefongespräch ist somit der Proxy-Server nicht mehr notwendig. Die Gesprächspartner senden sich ihre Daten direkt über das Internet zu.

VoIP-Software, die das SIP-Protokoll verwenden:
- X-Lite
- Jajah
- Gizmo

7.3.3. VoIP-Software

Für diese Arbeit wäre eine VoIP-Software geeignet, der man lediglich die IP-Adresse des Gesprächpartners übergeben kann. Eine aufwändige Adressübersetzung erübrigt sich dann.
Von den oben genanten VoIP-Programmen unterstützen dies folgende Programme:
- X-Lite
- NetMeeting

7.3.4. Datentransport

Ist mit Hilfe eines der oben beschriebenen Protokolle ein Verbindung aufgebaut worden, werden die Daten bei der Übertragung der Sprachsignale in viele kleine Pakete aufgeteilt. Der eigentliche Transport der Daten erfolgt über das Real-Time-Transport-Protocol (RTP), gesteuert durch das Real-Time-Transport-Control-Protocol (RTCP). RTP verwendet zur Übertragung in der Regel das User Datagram Protocol (UDP). UDP kommt zum Einsatz, da es ein minimales, verbindungsloses Netzwerkprotokoll ist, das nicht auf Zuverlässigkeit ausgelegt wurde wie beispielsweise das Transmission Control Protocol (TCP). Bei UDP wird der Empfang der Sprachpakete nicht bestätigt, es bestehet also keine Übertragungsgarantie. Dafür wird aber der Datenfluss auch nicht verzögert.

7.3.5. Nach dem Datentransport

Beim Empfänger werden die Pakete zunächst in einem Puffer zwischengespeichert. Von diesem Puffer werden sie zum Puffer des Mikrocontrollers (RAM-Speicher) übertragen. Wenn dieser Puffer voll ist, sollte ein Interrupt ausgelöst werden, der diese Daten an den PWM-Ausgang zum analogisieren schickt.

8. Fazit

Die große Motivation dieser Arbeit war, alte und neue Technik miteinander zu verbinden. Das alte Wählscheibentelefon und der moderne Mikrocontroller können problemlos miteinander kommunizieren und aufeinander reagieren. Entstanden ist ein Hardware-Interface, das für das Telefonieren über das Internet gerüstet ist.

Einen großen Teil dieser Arbeit stellte die Signalverarbeitung dar. Das Telefon erzeugt ein spezielles Signal, das es galt für den Mikrocontroller verständlich zu machen.

Die Kommunikation zwischen Mikrocontroller und PC funktioniert über USB. Da der PC keinen Treiber erhalten hat, ist es noch nicht möglich über das Internet zu telefonieren.

Mit dem funktionierenden Hardware-Interface ist es auf jeden Fall nicht mehr schwierig eine VoIP-Telefonie zu ermöglichen.

A Beigefügte CD-ROM

A.1. Inhalt der beigefügten CD-ROM

Diplomtheorie

Diplom_Altstaedter.pdf

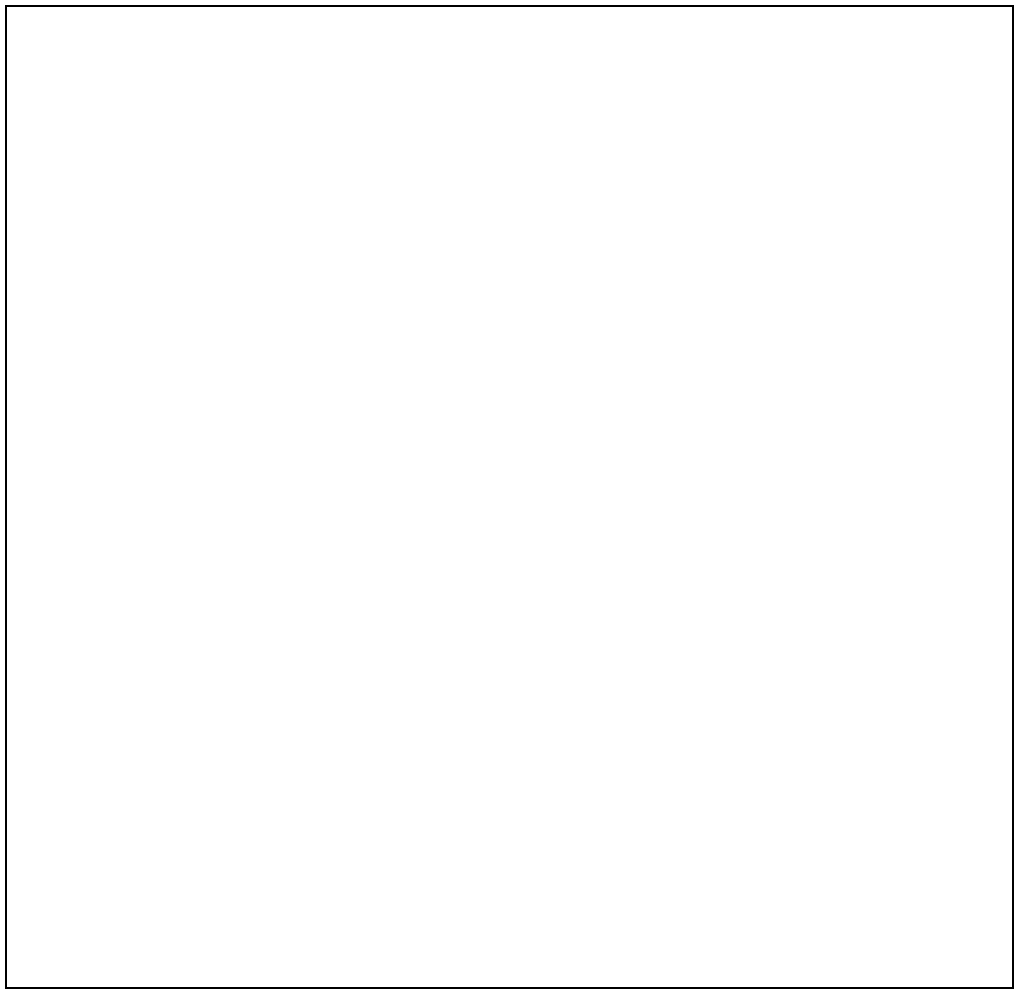
Programme

AVRDude
HyperTerminal
MProg
WinAVR

Quelltext

a2d.c
a2d.h
avrlibdefs.h
avrlibtypes.h
avrproj_make
buffer.c
buffer.h
cmdline.c
cmdline.h
cmdlineconf.h
cmdlinetest.c
cmdlinetest.hex
global.h
makefile
pwmsw.c
pwmsw.h
rprintf.c
rprintf.h
teltools.c
teltools.h
timer.c
timer.h
uart.c
uart.h
voip.bat
vt100.c
vt100.h

A.2. CD-ROM



B Lizenz

Der Inhalt dieser Arbeit ist unter einem **„Creative Commons Namensnennung-NichtKommerziell-Weitergabe unter gleichen Bedingungen 2.0 Germany“** Lizenzvertrag lizenziert. Um die Lizenz anzusehen, gehen Sie bitte zu http://creativecommons.org/licenses/by-nc-sa/2.0/de/.

Im Folgenden ist eine Zusammenfassung des Lizenzvertrages in allgemeinverständlicher Sprache angegeben:

creative commons

COMMONS DEED

Namensnennung-NichtKommerziell-Weitergabe unter gleichen Bedingungen 2.0 Deutschland

Sie dürfen:

- den Inhalt vervielfältigen, verbreiten und öffentlich aufführen
- Bearbeitungen anfertigen

Zu den folgenden Bedingungen:

Namensnennung. Sie müssen den Namen des Autors/Rechtsinhabers nennen.

Keine kommerzielle Nutzung. Dieser Inhalt darf nicht für kommerzielle Zwecke verwendet werden.

Weitergabe unter gleichen Bedingungen. Wenn Sie diesen Inhalt bearbeiten oder in anderer Weise umgestalten, verändern oder als Grundlage für einen anderen Inhalt verwenden, dann dürfen Sie den neu entstandenen Inhalt nur unter Verwendung identischer Lizenzbedingungen weitergeben.

- Im Falle einer Verbreitung müssen Sie anderen die Lizenzbedingungen, unter die dieser Inhalt fällt, mitteilen.
- Jede dieser Bedingungen kann nach schriftlicher Einwilligung des Rechtsinhabers aufgehoben werden.

Die gesetzlichen Schranken des Urheberrechts bleiben hiervon unberührt.

C Abbildungsverzeichnis

Abbildung 2.1.: Das Wählscheibentelefon 9
Abbildung 2.2.: Telefonsignaländerung beim Anrufen 10
Abbildung 2.3.: Telefonsignaländerung beim Angerufenwerden 11
Abbildung 2.4.: Der Mikrocontroller ATmega8 von der Firma Atmel 13
Abbildung 2.5.: Pin Konfiguration des ATmega8 [ATmega8] 13

Abbildung 3.1.: Lochrasterplatine 1 17
Abbildung 3.2.: Bestückungsplan von Platine 1 18
Abbildung 3.3.: Schaltplan von Platine 1 19
Abbildung 3.4.: Filter-Kurvenverlauf 24
Abbildung 3.5.: ISP-Programmieradapter 25
Abbildung 3.6.: Modemkabel mit Sub-D-Stecker 26
Abbildung 3.7.: USB-Interfacemodul DLP2232M 27
Abbildung 3.8.: Das Programm MProg 28
Abbildung 3.9.: Lochrasterplatine 2 30
Abbildung 3.10.: Bestückungsplan von Platine 2 31
Abbildung 3.11.: Schaltplan von Platine 2 32
Abbildung 3.12.: Ursprungsform Telefonsignal 35
Abbildung 3.13.: Signale an den Eingängen 3 (schwarz) und 2 (rot) 36
Abbildung 3.14.: reines Signal nach Wählimpuls-Schaltung 36
Abbildung 3.15.: Signal nach Tiefpassfilter 37
Abbildung 3.16.: Signale an den Eingängen 5 (schwarz) und 6 (rot) 38
Abbildung 3.17.: Ausgangssignal des Operationsverstärkers 38
Abbildung 3.18.: Beide Lochrasterplatinen sind miteinander verbunden 40

Abbildung 5.1.: PWM-Rechtecksignal 66
Abbildung 5.2.: Nicht invertierende PWM mit Tastverhältnis 50% 67
Abbildung 5.3.: Nicht invertierende PWM mit Vergleichswert 68
Abbildung 5.4.: Zählen der externen Interrupts 75

Abbildung 6.1.: Terminal-Fenster beim Anrufen 81
Abbildung 6.2.: Terminal-Fenster beim Angerufenwerden 82

D Glossar

ALU Arithmetisch-Logische Einheit

Eine arithmetisch-logische Einheit ist ein elektronisches Rechenwerk, welches in Prozessoren zum Einsatz kommt.

Batch-Datei

Stapeldatei mit einer Folge (= einem Stapel, Stapelverarbeitung) von Anweisungen oder Befehlen, die das Betriebssystem der Reihe nach abarbeitet.

Baudrate

Die Baudrate ist die europäische Bezeichnung für bps, d.h. Bits pro Sekunde, die Maßzahl für die Geschwindigkeit der Übertragung von digitalen Daten.

Impulswahlverfahren

Durch das Abheben des Telefonhörers beim analogen Endgerät wird zur Vermittlungsstelle eine Stromschleife geschlossen und von der Vermittlungsstelle der Wähl-Ton zum Teilnehmer gesendet. Das Betätigen des Nummernschalters unterbricht diese Schleife entsprechend der gewählten Ziffer. Bei Wahl der Ziffer 1 einmal, bei Ziffer 2 zweimal, ... bei Ziffer 0 zehnmal. Im Telefonhörer ist dies bei manchen Telefonen als eine Folge von Knacksern zu hören. Die gewählten Ziffern werden auf diese Weise in Wählimpulse umgesetzt. Sobald eine etwas längere Pause folgt, wird auf die nächste Ziffer gewartet. Bei alten analogen Telefonen ist so eine Impulswahl auch durch geschicktes Betätigen des Gabelumschalters möglich. [WIKI]

ISP In System Programmable

Das bedeutet, dass der Mikrocontroller in der Schaltung programmiert werden kann. Wenn ein Mikrocontroller diesen Vorteil nicht hat, dann muss man ihm zum Programmieren aus der Schaltung entfernen und in einem speziellen Programmiergerät programmieren.

Komparator

Ein Komparator ist ein Operationsverstärker, der ohne Gegenkopplung betrieben wird. Er vergleicht zwei Eingangssignale und gibt das Vergleichsergebnis am Ausgang aus: Wenn die Spannung am nicht-invertierenden Eingang höher ist als die Spannung am invertierenden Eingang so nähert sich die Ausgangsspannung der positiven Versorgungsspannung. Bei umgekehrten Verhältnissen geht die Ausgangsspannung gegen die negative Versorgungsspannung. [UniP]

NPN-Transistor

Ein Transistor ist ein elektronisches Halbleiterbauelement das zum Schalten und Verstärken von elektrischem Strom verwendet wird. Transistoren haben drei Anschlüsse. Mit der Bezeichnung npn-Transistor bezeichnet man einen Transistor, der die Schichtfolge n-dotiert (-), p-dotiert (+) und n-dotiert (-) auf weist. Entsprechend steht pnp-Transistor für die Schichtfolge p-dotiert (+), n-dotiert (-) und p-dotiert (+). Aus diesen drei Schichtfolgen ergeben sich auch die drei Anschlüsse, die ein Transistor in der Regel hat. Den mittleren Anschluss bezeichnet man als Basis (B) und die beiden äußeren Anschlüsse als Kollektor (C) und Emitter (E). Der Basisanschluss (B) dient bei der Verstärkerschaltung als Steueranschluss. So bewirkt ein relativer schwacher Strom an der Basis (B) die Steuerung des Stromflusses zwischen Kollektor und Emmiter. [UniP]

Operationsverstärker

Ein Operationsverstärker (Abk. OP, OPV, OV, OpAmp) besitzt einen invertierenden (-) und einen nichtinvertierenden Eingang (+), sowie einen Ausgang. Außerdem muss er eine Betriebsspannung erhalten.
Ohne jegliche zusätzliche Beschaltung liegt am Ausgang die volle positive oder negative Betriebsspannung an, abhängig von den anliegenden Spannungen an den Eingängen. Liegt am invertierenden Eingang eine höhere Spannung als am nichtinvertierenden, liegt der Ausgang auf negativer Betriebsspannung. Ist die Eingangsspannung genau anders herum, kann am Ausgang die positive maximale Spannung gemessen werden. Dies liegt daran, dass durch den Differenzverstärker und die anschließende große Verstärkung schon eine sehr kleine Spannungsdifferenz ausreicht, um den OP kippen zu lassen. In dieser Betriebsart arbeitet der Operationsverstärker als Komparator.
Durch unterschiedliche Außenbeschaltungen des Operationsverstärkers lassen sich die unterschiedlichsten Funktionen realisieren, beispielsweise Integrator und Differenzierer, Addierer und Subtrahierer oder auch einfach nur Verstärkerschaltungen mit einem vorher bestimmbaren Eingangs- und Ausgangsspannungsbereich. Diese Schaltungen werden auch heute noch verwendet, um beispielsweise Regler, Schmitt-Trigger und Filter wie Hochpass oder Tiefpass aufzubauen. [WIKI]

Potentiometer

Potentiometer sind regelbare Widerstände. Mit einem Drehknopf oder Schraubenzieher lässt sich der elektrische Widerstandswert verändern.

Pull-Up, Pull-Down

Das sind Widerstände, die die Pins eines Mikrocntrollers auf VCC (Pull-Up), oder auf GND (Pull-Down) ziehen. [ELKO]

Pulsweitenmodulation (PWM)

Die Pulsweitenmodulation (PWM) ist eine Modulationsart bei der eine technische Größe (z. B. elektrischer Strom) zwischen zwei Werten wechselt. Dabei wird das Tastverhältnis bei konstanter Frequenz moduliert. Ein PWM Signal wird allgemein über einen Tiefpass demoduliert. [WIKI]

Quarz / Schwingquarz

Eine Kristallstruktur, die sich durch elektrische Felder zu mechanischen Schwingungen anregen lässt. Der Quarz verhält sich elektrisch wie ein LCR-Glied mit parallel geschaltetem Kondensator und weist somit eine scharf definierte und stabile Parallel- und Serienresonanz auf. Der Quarz eignet sich daher besonders als Referenz für Schwingkreise, etwa in Taktgebern. [DesElek]

RC-Glied / RC-Filter

Eine Kombination aus Widerstand (R) und Kondensator (C), die einen als Tiefpass bezeichneten Filter bilden. Dieser Filter lässt nur Frequenzen unterhalb der Grenzfrequenz ungeschwächt passieren. Gebräuchlich sind solche Filter in der Elektronik, können aber auch in anderen Bereichen, wie zum Beispiel Mechanik, Akustik oder Hydraulik vorkommen. [Lexi]

Relais

Relais sind elektromagnetisch betätigte Schaltelemente mit geringer Schaltleistung. Man verwendet Relais für Steuerung und Regelungen. Ein Relais besteht aus einer Spule mit Eisenkern. Wird die Spule mit Strom durchflossen, so bewirkt das sich bildende magnetische Feld, dass sich Kontakte öffnen und schließen. [ELKO]
Ein monostabiles Relais fällt, nachdem der Strom zum Betätigen des Relais abgeschaltet wird, zurück in die Ausgangsstellung. [WIKI]

Schmitt-Trigger

Der Schmitt-Trigger ist ein analoger Komparator mit Mitkopplung. Er arbeitet also als Vergleicher für zwei analoge Spannungen. Er funktioniert als Schwellwertschalter. Durch die Mitkopplung besitzt er im Gegensatz zum Komparator jedoch unterschiedliche Ein- und Ausschaltschwellen, die um den Hysterese genannten Wert auseinanderliegen. [WIKI]

Spannungsteiler

Ein Spannungsteiler besteht im Regelfall aus zwei Widerständen, an denen sich die Gesamtspannung in zwei Teilspannungen aufteilt. Spannungsteiler werden verwendet, um Arbeitspunkte (Spannungsverhältnisse) an aktiven Bauelemente einzustellen. Hauptsächlich werden mit einem Spannungsteiler Spannungspotentiale erzeugt, die geringer als die Gesamtspannung sind. [ELKO]

Spikes und Glitches

Spikes sind kurzzeitig unerwünschte Signalwechsel auf einer einzelnen Signalleitung, während man unter Glitches unkorrekte Daten in einem Datenbus versteht, die aufgrund von Laufzeitunterschieden in der kombinatorischen Logik entstehen. [AnSch]

Status Register (SREG)

Das Status Register enthält Informationen über die Ergebnisse der meisten kürzlich ausgeführten arithmetischen Instruktionen. Diese Informationen können für den Programmfluss von Bedeutung sein, um zusätzliche Funktionen auszuführen.

Tantal-Elektrolytkondensator

Der Tantal-Elektrolytkondensator ist ein Elektrolytkondensator, dessen feste Elektrode aus Tantal besteht. Die Tantal-Elektroden werden je nach Bauform aus Folien gewickelt oder aus Pulvermaterial gesintert. [WIKI]

Tastverhältnis

Das Tastverhältnis (auch Tastgrad, engl. Duty cycle) gibt das Verhältnis der Länge des eingeschalteten Zustandes zur Periodendauer bei einem Rechteckssignal an.

Tiefpassfilter

Der Tiefpassfilter ist der in der Messtechnik am häufigsten benutzte Filter. Wie der Name schon sagt, lässt der Tiefpassfilter tieffrequente Signale passieren und filtert höherfrequente Signale aus. Frequenzen unter der Grenzfrequenz gelten als durchgelassene Frequenzen. Die vier häufigsten Tiefpassfilter-Typen sind: kritische Dämpfung, Bessel, Butterworth und Tschebyscheff. [Quali]

Wechselschalter

Ein Wechselschalter ist ein Schalter mit drei Anschlüssen, die an den mittleren Kontakt angeschlossene Leitung wird je nach Schalterstellung mit dem einen oder anderen Schalteranschluss verbunden.

E Quellenverzeichnis / Literaturverzeichnis

[AnSch] Homepage von Andreas Schwope
http://www.andreas-schwope.de/ASIC_s/Glossar/body_glossar.html

[ATmega8] Datenblatt von ATmega8
http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf

[ATMEL] Die offizielle Website von Atmel
http://www.atmel.com

[AVRdude] Dokumentation des AVRDUDE-Programms
C:\Programme\AVRdude\doc\avrdude-4.3.0\avrdude.pdf

[AVRfreaks] Die inoffizielle AVR Website von AVR Freaks
http://www.avrfreaks.com

[AVRGCC] AVR-GCC-Tutorial
http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial

[AVRrisc] Wolfgang Trampert, 2003
AVR-RISC Mikrocontroller, 2. Auflage
Franzis', ISBN 3-7723-5476-9

[AVR335] AVR335: Digital Sound Recorder with AVR and Serial DataFlash
http://www.atmel.com/dyn/resources/prod_documents/doc1456.pdf

[DesElek] Design&Elektronik
http://www.design-elektronik.de/elex

[ElekP] ELEKTRONIK PROJEKT
http://www.elektronik-projekt.de

[ELKO] Das Elektronik Kompendium
http://www.elektronik-kompendium.de

[Elektro] Horst Elschner, Albrecht Möschwitzer, 1992
Einführung in die Elektrotechnik-Elektronik, 3. Auflage
Verlag Technik GmbH Berlin, ISBN 3-341-00835-7

[Hess] Wolfgang Hess, 1993
Digitale Filter, 2. Auflage
Teubner Studienbücher Stuttgart, ISBN 3-519-16121-4

[Lexi] Lexikona
http://www.lexikona.de

[LM324] Datenblatt von vierfach Universaloperationsverstärker LM324
http://pdf1.alldatasheet.com/datasheet-pdf/view/11666/ONSEMI/LM324.html

[L78S05] Datenblatt des 2A Spannungsreglers L78S05CV
http://pdf1.alldatasheet.com/datasheet-pdf/view/22636/STMICROELECTRONICS/L78S05CV.html

[Mikro] Mikrocontroller Internetseite
http://www.mikrocontroller.net

[OPA2340] Datenblatt des zweifach einzelversorgten Rail-to-Rail Operationsverstärker OPA2340PA
http://pdf1.alldatasheet.com/datasheet-pdf/view/56765/BURR-BROWN/OPA2340PA.html

[Quali] Lexikon von Qualitätsmanagement
http://www.quality.de/lexikon

[Robo] Roboter NETZ Wissen, Roboter, Elektronik Mikrocontroller
http://www.roboternetz.de/wissen

[RoboF] Roboter NETZ Forum, Roboter, Elektronik Mikrocontroller
http://www.roboternetz.de

[Schule] Winfield Hill, Paul Horowitz, 2002
Die hohe Schule der Elektronik, Teil1, Analogtechnik, 7. Auflage
Elektor-Verlag, ISBN 3-895-76024-2

[Signal] Samuel D. Stearns, Don R. Hush, 1994
Digitale Verarbeitung analoger Signale, 6. Auflage
R. Oldenbourg Verlag GmbH München, ISBN 3-486-22027-6

[Telt] Teltarif.de
http://www.teltarif.de/i/voip.html

[UniP] Lexikon von www.uni-protokolle.de
http://www.uni-protokolle.de/Lexikon

[VirUni] Virtual University
http://www.virtualuniversity.ch/telekom/voip/

[WIKI] WIKIPEDIA
http://de.wikipedia.org